VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Sledování aktuálního stavu v průběhu soutěží

Actual State Monitoring During Competition

2017 Filip Maceček

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Filip Maceček**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Sledování aktuálního stavu v průběhu soutěží
Actual State Monitoring During Competition

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je navrhnout a realizovat komplexní řešení systému pro sledování stavu všech her při různých soutěží typu jeden proti jednomu. Vyberte vhodné moduly pro přenos informací mezi účastníky soutěže a monitorovacími počítači. Dále pak navrhněte potřebné programové řešení pro ukládání stavu jednotlivých soutěží a grafickou prezentaci průběžných výsledků.

1. Popište požadavky kladené na systém sledování průběžného stavu soutěže.
2. Vyberte a otestujte vhodné moduly pro přenos informací mezi hráči a počítačem. Otestujte zejména dosah signálu a spolehlivost přenosu informací. Vyberte nejvhodnější modul nebo moduly.
3. Navrhněte architekturu celého systému a vyberte vhodné technologie.
4. Implementujte systém dle navrženého modelu.
5. Proveďte testování výsledné aplikace, vyhodnoťte stabilitu a spolehlivost systému.

Seznam doporučené odborné literatury:

[1] Shaw, Zed A. Learn Python the Hard Way: A Very Simple Introduction to the Terrifyingly Beautiful World of Computers and Code. Addison-Wesley, 2013.
[2] Stones, Richard, et al. Linux: začínáme programovat. Computer Press, 2000.
[3] Raspberry Pi: http://www.raspberrypi.org/
[4] Pyserial: http://pyserial.sourceforge.net/

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Petr Olivka, Ph.D.**

Datum zadání: 01.09.2015

Datum odevzdání: 28.04.2017

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 1. dubna 2017 .....

Rád bych poděkoval panu doktorovi Petru Olivkovi za odbornou pomoc a konzultaci při vytváření této bakalářské práce a také panu Richardu Šnapkovi za námět, který dal práci vzniknout.

Abstrakt

Cílem práce je navrhnout a implementovat komplexní řešení pro sledování stavu různých soutěží typu jeden proti jednomu. Práce zahrnuje sestrojení tří prototypů a testování jejich dosahu a výdrže na baterii. Dále také obsahuje návrh a implementaci cloudového řešení pro odezvu na události z výše zmíněných prototypů, jejich zpracování, zobrazení atp. Prototypy byly sestrojeny za použití volně dostupných modulů pro bezdrátovou komunikaci (Bluetooth, WiFi, 440 MHz rádio). Cloudové řešení bylo postaveno na technologii ASP .NET s databázovou vrstvou SQL Server a jako klient pro zobrazování dat webová aplikace postavena na frameworku Angular 2 a jQuery. Vytvořený systém umožňuje 100% spolehlivý bezdrátový přenos informace o stavu na vzdálenost až 100 m při výdrži baterie při vysílání 20 hodin a odezvy systému v průměru 121 ms.

Klíčová slova: WiFi, Bluetooth, radio, battery, prototyp, Arduino, microchip, Raspberry Pi, HC-05, NodeMCU, ASP .NET, .NET Framework, MVC, SignalR, Entity Framework, SQL Server, Angular 2, jQuery, HTML, JavaScript, Highcharts, Less, CSS

Abstract

The target of this work is to design and implement complex solution for observing state of various contests. The work contains construction of three prototypes for wireless communication. It also contains design and implementation of cloud solution, which response to events from those prototypes, data processing, visualization etc. The prototypes were build using freely available modules for wireless communication (Bluetooth, WiFi, 440mHz radio). Cloud solution was developed using ASP .NET and SQL Server and client for data visualization as web application using Angular 2 and jQuery. Implemented system is able to transfer and process information with one-hundred success rate over 100 meters while maintaining battery life for circa 20 hours and with average system response of 121 ms.

Key Words: WiFi, Bluetooth, radio, battery, prototype, Arduino, microchip, Raspberry Pi, HC-05, NodeMCU, ASP .NET, .NET Framework, MVC, SignalR, Entity Framework, SQL Server, Angular 2, jQuery, HTML, JavaScript, Highcharts, Less, CSS

Obsah

Seznam použitých zkratek a symbolů

AJAX	–	Asynchronous JavaScript and XML
CSS	–	Cascading Style Sheets
EF	–	Entity Framework
GUID	–	Global Unique Identifier
HTML	–	Hyper Text Markup Language
IoC	–	Inversion of Control
JS	–	Javascript
MVC	–	Model View Controller
RPi	–	Raspberry Pi
SPA	–	Single Page Application

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Mějme dva aktéry Alici a Boba, kteří jsou aktivními sportovci. Navzájem spolu soupeří v různých soutěžích, jako je například badminton či tenis. Při hře se mohou snadno dostat do situace, kdy si ani jeden z aktérů správně nevybavuje stav jejich skóre. Navíc při každé takové hře ztrácí pojem o dlouhodobých statistikách. Aby dokázali posoudit své dlouhodobé statistiky, museli by si pamatovat každou hru, což je takřka nemožné. I kdyby se jim toto podařilo, stále nemají prostředek k tomu, aby mohli vyčíslit, jakou měrou se každý z nich za určitý čas zlepšil nebo například k tomu určit, jak moc byly za nějaký časový úsek aktivní, co se počtu her týče. Tato bakalářská práce popisuje vývoj právě takového prostředku, který by Alici i Bobovi umožnil nejen ukládat statistiky svých her, ale také vizualizovat je v přehledných grafech a tabulkách, intuitivně, kdykoliv a kdekoliv pomocí internetového prohlížeče v jakémkoliv zařízení od mobilních po desktopové.

Prvním problémem je, jak umožnit aktérům říci systému, že se skóre změnilo. Například tlačítko, jejímž stisknutím by se skóre změnilo. Pro lehčí manipulaci by tlačítko mělo být bezdrátové. Dále je třeba někde skóre zobrazit. Nabízí se nějaké jednoduché zobrazovací zařízení, které dokáže systém ovládat, visící nad kurtem. Ideálně takové, na kterém by se v mezi čase mohli zobrazovat reklamy či jiný obsah. Jakákoliv LCD televize či monitor postačí. Skóre se navíc musí ukládat pro zobrazení v budoucnosti a navíc, posílat zájemcům informace o své změně. Tato práce popisuje návrh právě takového systému.

2 Sběr požadavků

Tato kapitola se zabývá určením požadavků na systém. Požadavky jsou zkonstruovány tak, aby systém nezávisel na konkrétním scénáři použití, jinými slovy chceme, abychom byly schopni systém přizpůsobit jakémukoliv scénáři bez větších zásahů. Můžeme si například představit, že do systému později budeme chtít zapojit i různé druhy zařízení, které se od sebe značně odlišují. Co když se například rozhodneme, že Alici a Bobovi umožníme počítat si skóre nejen ve svých oblíbených sportovištích, ale i mimo ně ze svých mobilních zařízeních? Kdybychom takto razantně změnili strategii a náš systém by byl připraven pouze na konkrétní scénář, jeho rozšíření by bylo nákladné a časově náročné.

2.1 Spolehlivost

Nechceme, aby se uživatelé museli po každé, kdy přičtou skóre, starat o to, jestli se skutečně přičetlo. Takové chování by prokázalo systém jako neužitečný. Kromě zřejmých opatření proti nespolehlivosti, jako je například ošetření různých výjimečných situací, které můžou v kódu systému nastat, musíme také při návrhu volit takové postupy, které nám spolehlivost zaručí.

2.2 Čas odezvy

Tento bod jde se spolehlivostí ruku v ruce. Systém musí odpovídat v rozumném čase. Nesmíme nechat uživatele čekat příliš dlouho na odpověď systému. Dlouhá odezva by navíc mohla vést k tomu, že hráč neví, jestli skóre skutečně přičetl, a tudíž přičte skóre znovu. Odezva v tomto případě znamená časový úsek mezi časem kdy hráč přičte skóre a mezi tím, než se mu přičtené skóre zobrazí na obrazovce. Ideální čas odezvy závisí na konkrétní soutěži, nicméně můžeme bezpečně říct, že čas delší než 1 vteřina je nepřijatelný.

2.3 Intuitivnost

Dalším důležitým aspektem našeho systému, chceme-li aby ho používalo co možná nejvíce lidí, je bezpochyby intuitivnost systému. To znamená, systém musí být jednoduchý a pohodlný na ovládání a toto se netýká pouze procesu změny skóre, ale také jednoduché orientace a ovládání v dalších přidružených aplikacích jako je například webový portál. Abychom tohoto aspektu dosáhli, založíme mechanizmus přičítání skóre na jednom tlačítku, které pomocí různých způsobů stisku může přičítat, opravovat a ukládat skóre. Odpadne nám tedy potřeba více tlačítek, a zařízení se zmenší. Známkou intuitivnosti v moderním systému může být také tzv. Single Page Application (SPA) webový portál, který se neaktualizuje s každým požadavkem na server, ale aktualizují se pouze jeho součásti.

2.4 Dostupnost

Velmi klíčovým požadavkem je dostupnost. Systém musí být dostupný odkudkoliv, v kterýkoliv čas.

2.5 Modularita

Jakýkoliv funkční blok našeho systému musí být nahraditelný za jakýkoliv jiný stejného druhu. Jinými slovy, každý funkční blok musí být schován za určitou úroveň abstrakce. Používáme-li technologii SQL Server pro databázovou vrstvu, nemělo by nám dělat problém tuto vrstvu nahradit například technologií MySQL. Úroveň abstrakce dělá celý systém testovatelný, protože jednotlivé abstrakce můžeme nahrazovat umělými implementacemi, a také škálovatelný.

2.6 Rozšiřitelnost

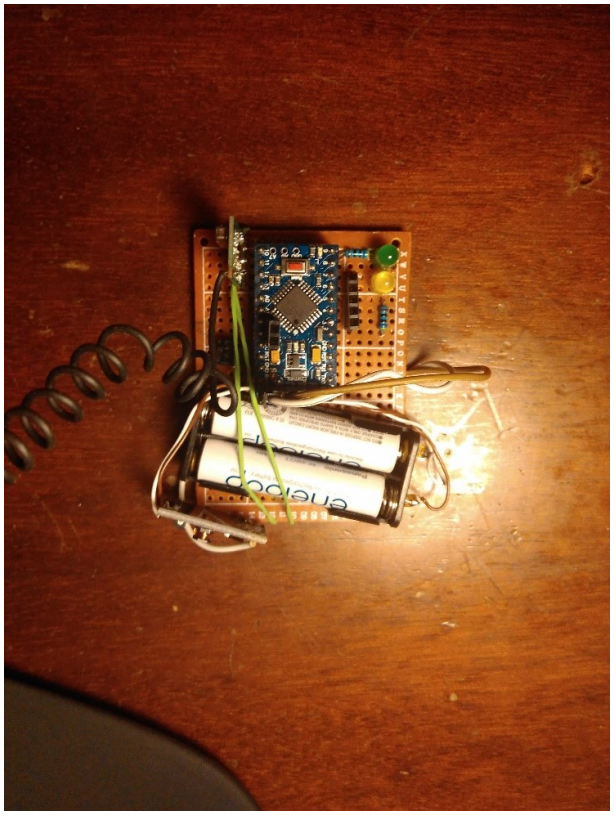
V neposlední řadě, náš systém musí být také jednoduše rozšiřitelný. To znamená návrh musí být takový, aby programátorovi umožnil jednoduché rozšíření o další funkcionalitu.

3 Prototypy ovladačů

Jak už bylo dříve řečeno, je třeba dát uživatelům způsob, jak přičítat skóre, případně opravovat, resetovat, ukládat atp. Toto ovládání je realizováno jednoduchými zařízeními, které kromě elektroniky pro komunikaci na dálku obsahují jediné tlačítko. Toto tlačítko umí dělat všechny zmíněné operace, které rozlišuje na základě gesta provedeného na tlačítku. Tato gesta si můžeme představit jako jednoduchý stisk, dlouhý stisk a dvojitý stisk stejně jako na klasické počítačové myši. Taková zařízení mohou mít různou podobu. Můžeme si například představit ovladač podobný televiznímu dálkovému ovládání s jedním tlačítkem, nebo například vkusný náramek, který po zmáčknutí funguje jako ovladač. Ovšem pro prototypovací účely zde bude elektronika umístěna na prototypové desky či nepájivá pole.

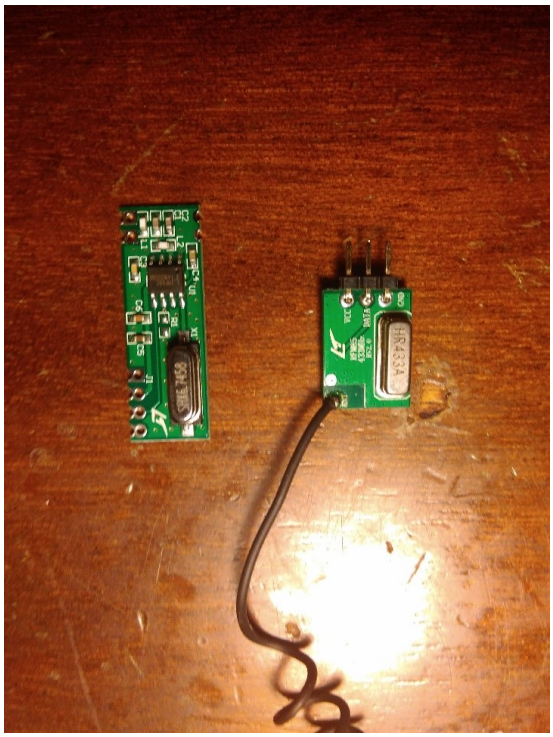
3.1 AM Rádio

První prototyp používá jako technologii pro přenos informace jednoduché rádio s amplitudovou modulací na frekvenci 433 MHz. Prototyp vysílače můžeme vidět na obrázku 1.



Obrázek 1: 433 MHz AM rádio prototyp

Jedná se o jednoduchý AM vysílač RFM85[1] (obrázek 2) od firmy Hoperf electronic připojený k desce mikrokontroleru Arduino[5]. Elektronika je pájena na prototypovou desku. Deska je napájena sadou AAA bateriemi připojenými přes regulátor napětí. Prototyp také obsahuje dvě LED diody pro signalizaci stavu pro snadnější vývoj, samotné tlačítko na ovládání a hlavičku pro připojení programátoru. Program do mikrokontroleru se dá nahrát přes typický USB – Serial převodník.



Obrázek 2: Přijímač RFM83C a vysílač RFM 85

Kód pro poslání informace je velmi jednoduchý. Použiji knihovnu třetí strany kompatibilní s modulem RFM85. Takových knihoven je dostupných mnoho, avšak pro moje účely jsem vybral knihovnu *RCSwitch*[2], a to především kvůli faktu, že na rozdíl od ostatních knihoven, pro *RCSwitch* existuje port pro RPi, jinými slovy jsem schopnen zachytávat data z vysílače na RPi stejným kódem. Práce s knihovnou *RCSwitch* je velmi jednoduchá. Nejprve inicializuji třídu *RCSwitch* dle výpisu 1.

```
RCSwitch            mySwitch = RCSwitch();

void setup(){
   // ..inicializacni kod
   mySwitch.enableTransmit(TRANSMIT_PIN);
   // ..dalsi inicializacni kod
}
```

Výpis 1: Inicializace třídy RCSwitch

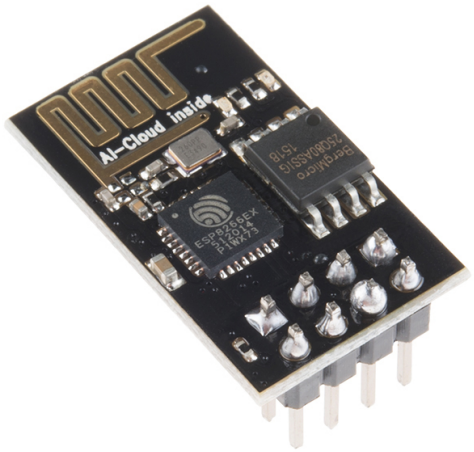
```
void send_gesture(int gesture)
{
   toggle_led();
   switch (gesture)
   {
   case SHORT_PRESS:
      // Posleme data.
      mySwitch.send(ID + gesture, 16);
      break;
   // .. atd.
}
```

Výpis 2: Poslání zprávy přes RCSwitch

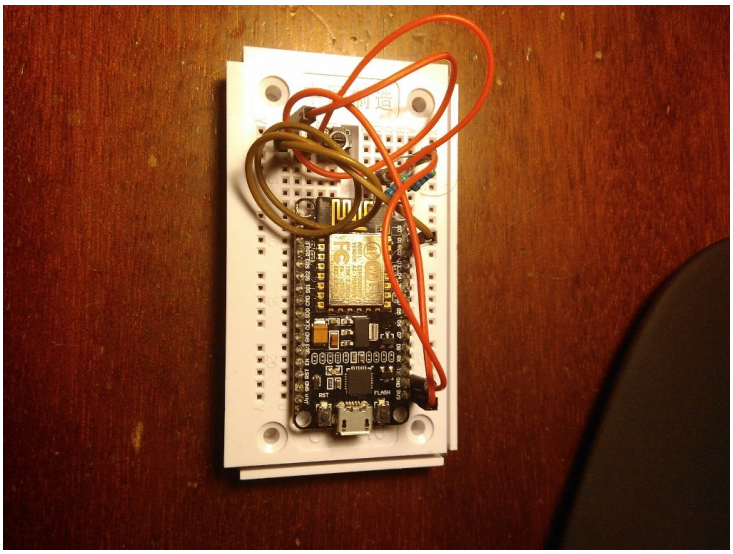
Ve výpisu 2 můžeme vidět příklad poslání informace. Všimněte si, že metodu „send“ volám se dvěma parametry. První je informace, která se skládá z ID zařízení pro rozpoznání, ze kterého zařízení data přišla a jaké gesto bylo rozpoznáno (pro připomenutí, pro intuitivnost používám pouze jedno tlačítko, na kterém rozpoznávám různá gesta) a druhým je informace, jak dlouhá jsou posílaná data. V mém případě jde o *unsigned integer*, který má na platformě Arduino 16 bitů.

3.2 NodeMCU

S příchodem WiFi modulu ESP8266[3] (obrázek 3) na trh přišla i revoluce v internetu věcí. Tento modul velikosti korunové mince nabízí relativně levnou platformu pro vývoj IoT aplikací. ESP8266 původně nebyl určen pro to, aby do něj uživatel nahrával svůj vlastní program. Jeho integrovaný procesor se měl pouze starat o konektivitu, zatímco jiný mikročip měl rozdávat modulu data. To proto, aby se ušetřil procesorový čas modulu. Programátoři se nicméně s tímto nespokojili a našli způsob, jak do modulu nahrát svůj vlastní program načež výrobci ESP8266 později vydali oficiální SDK. NodeMCU[4] zapouzdřuje ESP8266 do praktické vývojové desky s USB – Serial převodníkem.



Obrázek 3: Samostatný modul ESP8266



Obrázek 4: NodeMCU na nepájivém poli

Na obrázku 4 můžete vidět NodeMCU zapojeného do nepájivého pole. Zapojení zahrnuje tlačítko připojené k modulu pro ovládání hry, stejně jako u předchozího prototypu. Pro signalizaci použiji integrovanou diodu. Kód, kterým se NodeMCU připojí do sítě, je vidět na výpisu 3.

```
void setup() {
   // PINS
   pinMode(BUTTON_PIN, INPUT);
   pinMode(LED_BUILTIN, OUTPUT);
   LedOff();
   // Serial
   Serial.begin(SERIAL_SPEED);
   // WiFi
   WiFi.begin(WIFI_SSID, WIFI_PW);
   Serial.write("Connecting\n");
   wl_status_t stat = APConnect();

   if (stat != WL_CONNECTED) {
      Serial.write("AP Connection failed.\n");
      error();
   }

   Serial.write("Connected.\n");
}
```

Výpis 3: Připojení NodeMCU do sítě

Program nejprve nastaví piny procesoru a poté inicializuje sériový port pro zasílání hlášení (můžu tak vidět co se v programu děje, mám-li desku připojenou k počítači). Následuje pak už jen volání *WiFi.begin*, které jako parametry vložím název WiFi sítě a heslo k ní. Dále se ověří, jestli se podařilo modulu skutečně do sítě připojit. Přijímačem v tomto případě může být jakýkoliv bod v síti. Pro naše účely, řekněme, že máme někde v síti vystavený http end-point, který přičítá skóre. Jeho zavolání je ukázano ve výpisu 4.

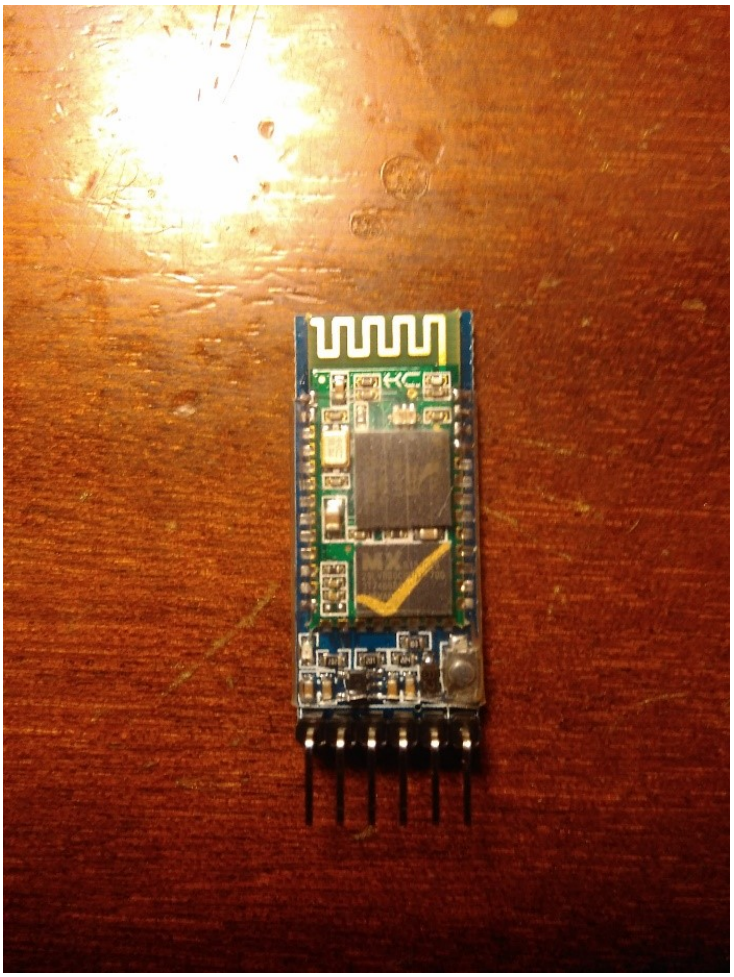
```
if (WiFi.status() != WL_CONNECTED) {
   error();
}
char url[250];
sprintf(url, "http://192.168.0.105/uScore.WebApi/api/sessionaction/
    incrementscore?clickerGuid=%s", GUID);
http.begin(url);-
http.addHeader("Content-Length", "0");
int code = http.POST(0, 0);
```

Výpis 4: Zavolání web api metody

Ověřím, zda jsem stále připojen k síti a po sléze do cílové URL přidáme parametr, kterým je ID našeho prototypu. Vytvořím *request*, přidám hlavičku s délkou obsahu 0 (kdybych to neudělal mohl by dojít k neočekávanému chování na straně serveru).

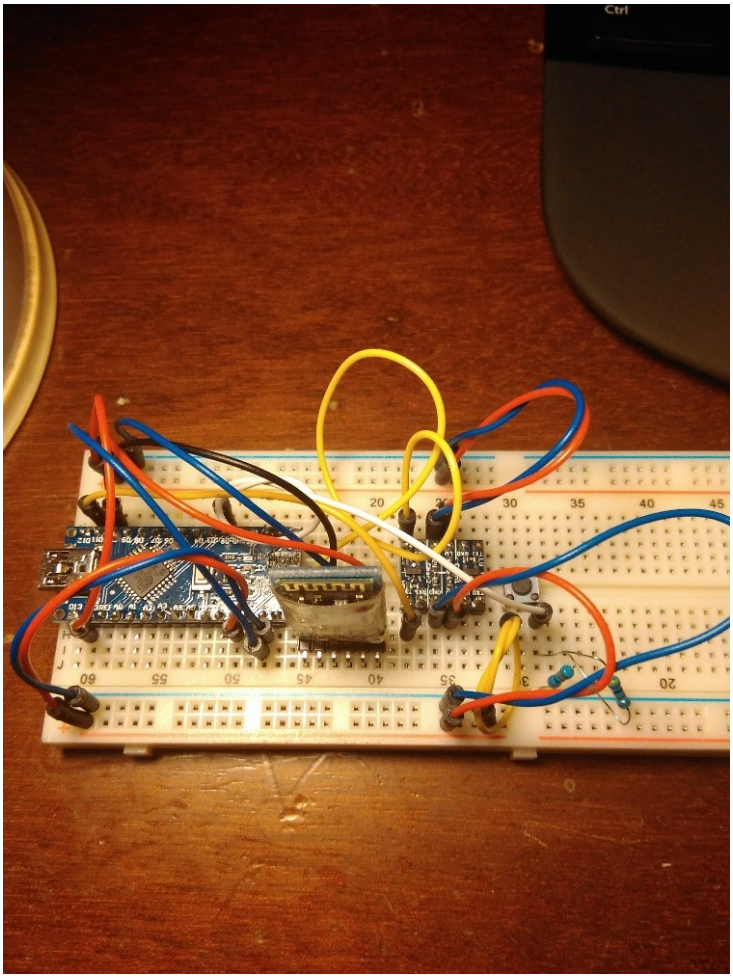
3.3 HC-05

HC-05 je modul pro bezdrátovou komunikaci přes technologii *Bluetooth* (obrázek 5). Je to nejdostupnější modul na českém trhu a je extrémně jednoduchý k použití. O veškeré párování se totiž stará samotný modul, a vše co uživatel dělá, je to, že přijímá zprávy, které modul zachytil, a naopak posílá zprávy do modulu, které má modul poslat spárovanému zařízení.



Obrázek 5: Modul HC-05

Stejně jako první prototyp, i tento připojím k Arduinu, přes které budu modulu posílat data, která má poslat. Před vysíláním musím modul spárovat s přijímačem. Tím může být jakékoliv zařízení, které oplývá Bluetooth technologií (např. mobilní telefon). Do zapojení samozřejmě patří i tlačítko. Prototyp můžeme vidět na obrázku 6.



Obrázek 6: Prototyp s HC-05

Modul je k Arduinu připojen přes sériové rozhraní. Kód, kterým pošlu přes modul zprávu, je vidět ve výpisu 5. Podmínkou pro úspěšné poslaní je, že modul byl předem spárován s přijímačem.

```
if (Serial.available()) {
   Serial.write('Hello world!');
}
```

Výpis 5: Zaslání zprávy modulem HC-05

4 Testování prototypů

V předchozích kapitolách jsem stručně představil prototypy ovladačů k ovládání hry. Abych splnil požadavky, zejména spolehlivost a odezvu, je třeba prototypy podrobit testům, které prozkouší jejich bateriovou výdrž a jejich dosah.

4.1 Dosah

V této kapitole se pokusím přiblížit měření dosahu prototypů.

4.1.1 Pomůcky

Pro měření byly použité tyto přijímače:

Prototyp	Přijímač
AM Rádio	Modul RFM83C
NodeMCU	TP-Link TL-WR841N (Běžný domácí WiFi router)
HC-05	Qualcomm Atheros QCA9565 (Bluetooth čip v typickém laptopu)

Tabulka 1: Použité přijímače

4.1.2 Cíl měření

Cílem měření je ověřit spolehlivost přijetí zprávy ve vzdálenostech 10, 15, 20, 40, 60, 80 metrů na přímou viditelnost s přijímačem bez překážek u všech prototypů.

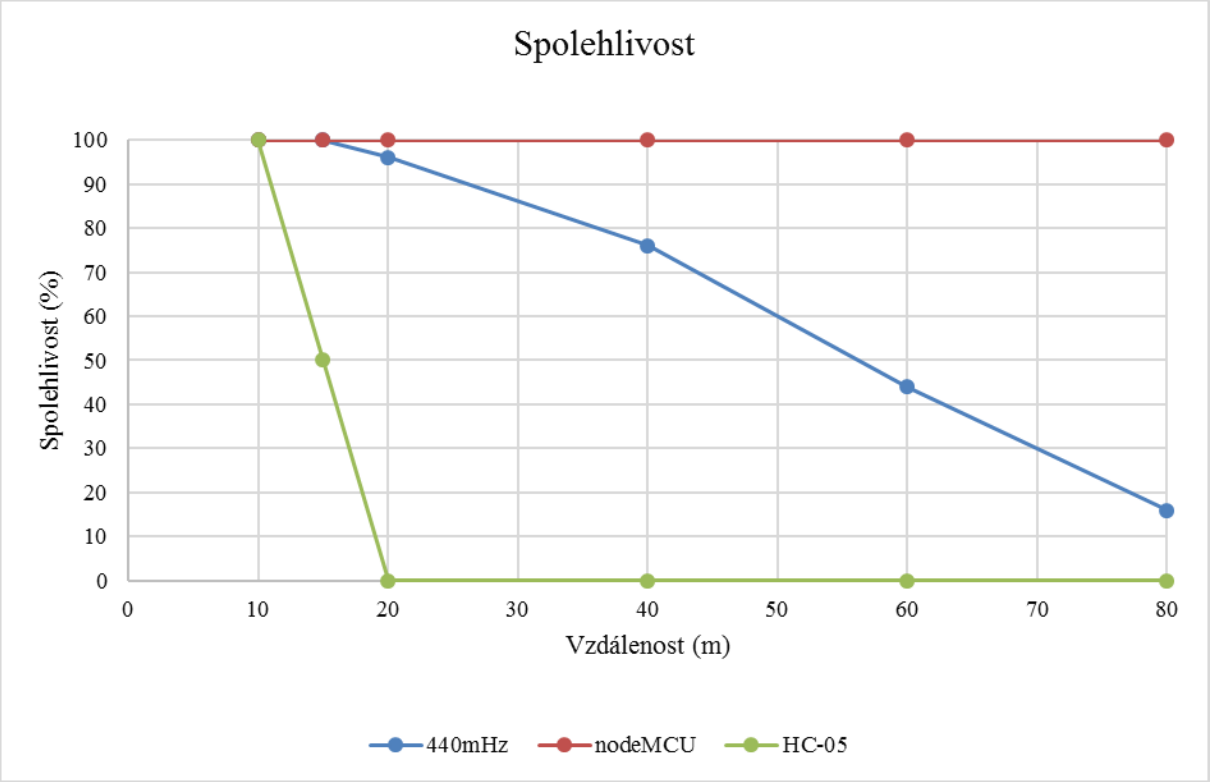
4.1.3 Postup měření

Mám otevřený prostor a v něm bod, ve kterém je přijímač pro daný typ prototypu. Odstupuji od přijímače a pokaždé když dojdu k cílové vzdálenosti od něj stisknu tlačítko vysílače v pěti vteřinových intervalech celkem třicetkrát a před další vzdálenosti minutu vyčkám. Po poslední měřené vzdálenosti přečtu z přijímače data a podle časů přijetí jednotlivých zpráv určím kolik zpráv bylo přijato na kterou vzdálenost. Po takovémto rozlišení dat pak vyčíslím spolehlivost v jednotlivých vzdálenostech.

4.1.4 Naměřené hodnoty

Vzdálenost (m)	440mHz (%)	NodeMCU (%)	HC-05 (%)
10	100	100	100
15	100	100	50
20	96	100	0
40	76	100	0
60	44	100	0
80	16	100	0

Tabulka 2: Tabulka spolehlivosti jednotlivých prototypů



Obrázek 7: Graf spolehlivosti prototypů

4.1.5 Závěr měření

Připomenu jeden z hlavních požadavků na systém, a sice na spolehlivost. Chci, aby si hráč nelámal hlavu tím, jestli se skóre přičetlo či nikoliv. Proto musím zvolit takový modul, který co nejspolehlivěji přenese stisk tlačítka. Abych závěr přizpůsobil reálnému světu, budu se odkazovat na tenisový kurt, který měří 23,78 metrů na délku a 10,97 metrů na šířku.

První prototyp postavený na jednoduchém AM rádiu má sice velký dosah, nicméně jeho spolehlivost se s narůstající vzdáleností od přijímače klesá. Konkrétně na 15 metrů máme 100% spolehlivost, avšak pokud by hráč chtěl přičíst skóre přijímači, které je na opačné straně kurtu, tedy necelých 24 metrů daleko, šance klesá pod 90 % což je nepřípustné.

Další prototyp založený na technologii Bluetooth bohužel taky nepřipadá v úvahu, byť přenos informace je na krátkou vzdálenost 100% avšak už při 15 metrech klesá na 50 % což v případě tenisového kurtu je nedostačující.

Nejlépe dopadl prototyp postavený na NodeMCU. 100% spolehlivost vykazuje na vzdálenost větší než 80 metrů, čímž je nejlepší volbou. Výhodu navíc vidím v tom, že může být připojen přímo do internetu a komunikovat tak s internetovou aplikací bez mezičlánků.

4.2 Výdrž na baterii

Bude následovat měření výdrže na baterii jednotlivých prototypů. Použité baterie jsou běžně dostupné a hojně používané.

4.2.1 Pomůcky

Multimetr LINI-T UT20B, baterie GP AA NiMH 2600mAh a Sony CR2032 220mAh.

4.2.2 Cíl měření

Cílem měření je změřit odběr jednotlivých prototypů v režimu čekání na stisk (režim Idle) a při vysílání. Dalším cílem je z naměřených hodnot spočítat celkovou výdrž v hodinách při použití 2 typů baterií a sice GP AA NiMH 2600mAh (tzv. *„tužková“*) a Sony CR2032 220mAh (tzv. *„knoflíková“*).

4.2.3 Postup měření

Použiju USB jako zdroj napětí. Z USB povede pouze napětí na vstupní napájení do prototypů, tedy nebudou zapojeny datové vodiče. Mezi 5 V uzel a prototyp připojím multimetr v režimu *ampérmetru*. Změřím proud vstupující do prototypu v režimu Idle (čekání na stisk tlačítka) a při soustavném vysílání. Následně spočítám podle hodnot vybraných baterií předpokládanou výdrž.

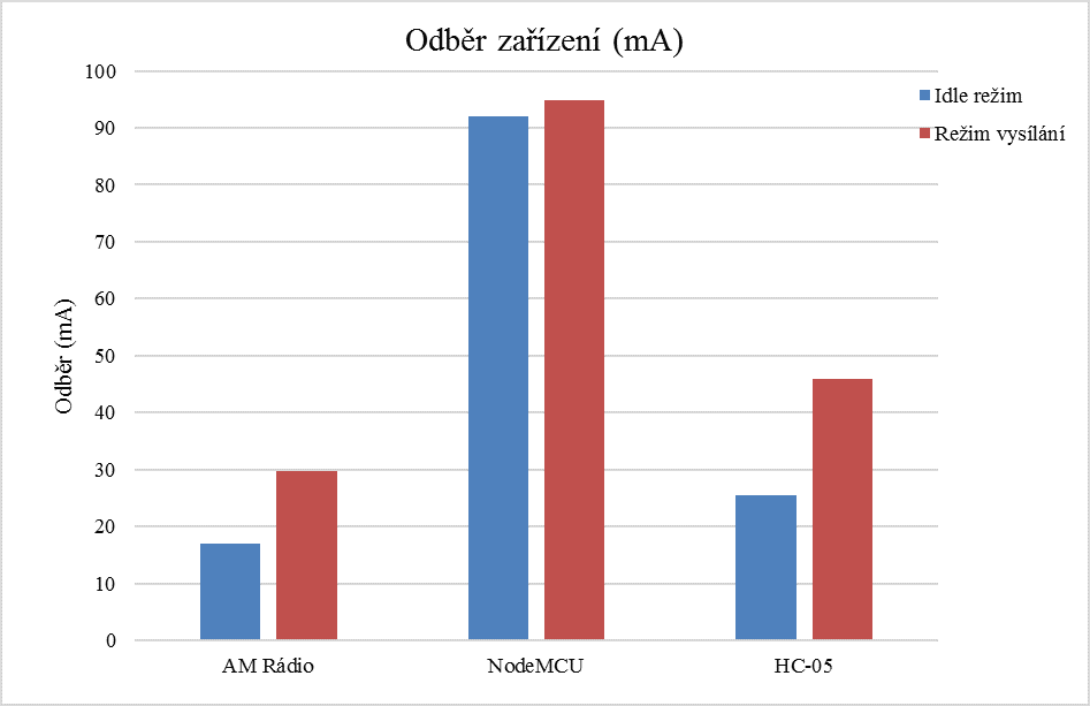
4.2.4 Naměřené hodnoty

Prototyp	AM Rádio (mA)	NodeMCU (mA)	HC-05 (mA)
Idle režim	17	92.1	25.5
Režim vysílání	29.7	94.9	46

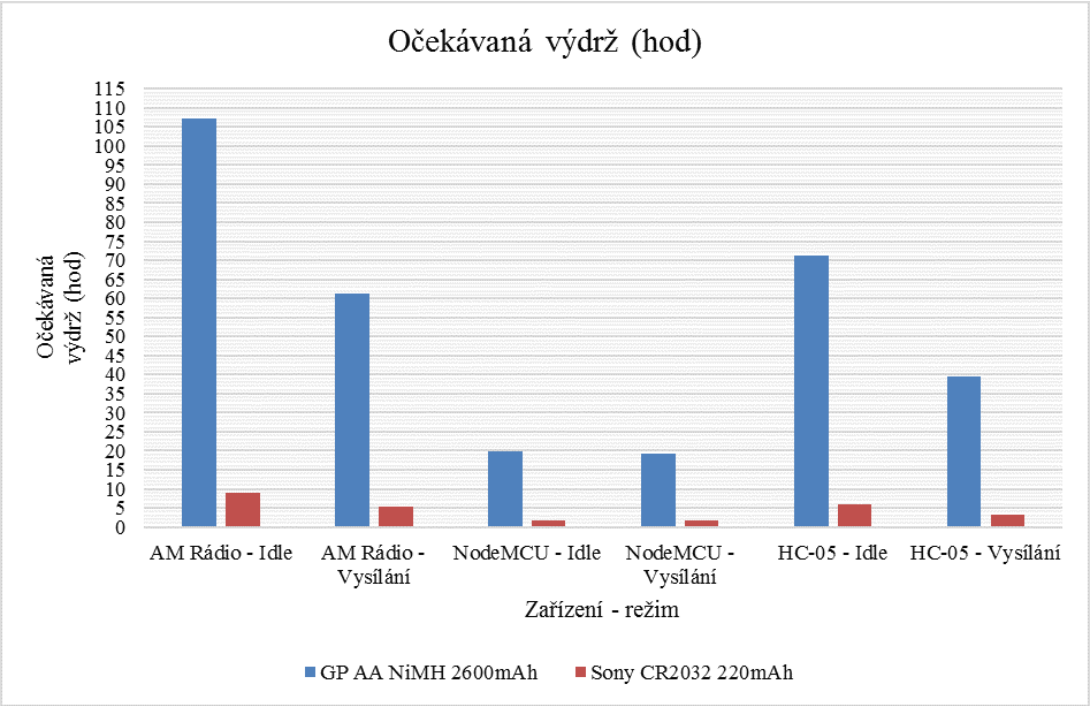
Tabulka 3: Naměřené odběry prototypů

Zařízení	GP AA NiMH 2600mAh (hod)	Sony CR2032 220mAh (hod)
AM Rádio - Idle	107.05882	9.05882
AM Rádio - Vysílání	61.27946	5.18519
NodeMCU - Idle	19.76113	1.6721
NodeMCU - Vysílání	19.17808	1.62276
HC-05 - Idle	71.37255	6.03922
HC-05 - Vysílání	39.56522	3.34783

Tabulka 4: Předpokládaný výdrž na baterii v hodinách



Obrázek 8: Graf odběru zařízení



Obrázek 9: Očekávaná výdrž

4.2.5 Závěr měření

Nejlépe dopadl AM Rádio prototyp s výdrží až 107 hodin v Idle režimu a až 61 hodin při vysílání na tužkovou baterku a 9 hodin v Idle režimu a 6 hodin při vysílání při užití knoflíkové baterie.

Dalším je HC-05 prototyp, jehož výdrž je 71 hodin v Idle režimu a 39 hodin v režimu vysílání s tužkovou baterií. Co se knoflíkové baterie týče očekávaná výdrž je 6 hodin v Idle režimu a 3 hodiny v režimu vysílání.

Nejhůře dopadl NodeMCU prototyp. S tužkovou baterií vydrží 19 hodin v Idle režimu i ve vysílání. S baterií knoflíkovou vydrží méně než 2 hodiny.

Je třeba dodat, že výdrž se dá optimalizovat mnoha způsoby. Až na NodeMCU, všechny prototypy používají pro řízení Arduino. Co se elektroniky týče, přebytečný odběr proudu vytváří právě tyto desky a jejich obvody, především USB-Serial převodníky, ale i LED diody. Tento „overhead“ má i NodeMCU. Další přebytečný odběr je v samotném programu. Všechny prototypy ověřují cyklicky stav tlačítka. Použitím režimů spánku a modifikací programu tak, aby reagoval na přerušení z pinu tlačítka, bych dokázal mnohem lépe odběr optimalizovat, především u NodeMCU. Tyto optimalizace ovšem platí jen v Idle režimu.

4.3 Závěr

Cílem předchozích kapitol bylo vybrat správný modul pro bezdrátové ovládání hry. I když NodeMCU nedopadl dobře při testování výdrže (necelé 2 hodiny při vysílání i v Idle režimu), je stále ideální volbou pro splnění požadavků na systém. NodeMCU zcela jistě splňuje spolehlivost. Dosah je až 100 metrů se 100% pravděpodobností na přenesení zprávy.

Modul také splňuje dostupnost. Vše, co potřebujeme k jeho provozu je WiFi síť, která je dnes nejrozšířenější médium pro bezdrátovou komunikaci.

Byť modul dopadl špatně při testech výdrže, je třeba zvážit, jak často bude zařízení v provozu a jak často bude vysílat. Jak jsem zmínil na závěru měření, jeho výdrž v Idle režimu se dá do značné míry optimalizovat a co se vysílání týče, potřebujeme posílat zprávy, jejichž poslání, za podmínky, že nebude problém se samotnou sítí, by nemělo trvat déle než vteřinu.

5 Návrh systému

5.1 Ukázka konceptu

Ještě než začnu popisovat celý systém, ukážu nejprve původní řešení, které bylo pro svá omezení v rámci rozšiřitelnosti, modularity a dostupnosti nakonec zavrhnuto. V dalších kapitole vysvětlím, proč je tomu tak.

Jedná se o aplikaci napsanou pro *Raspberry Pi*[7] (RPi). Raspberry Pi je jednodeskový počítač, založený na architektuře *ARM*. Jak takový počítač vypadáme, můžeme vidět na obrázku 10.

Obrázek 10: Raspberry Pi

Aplikace se starala o zachytávání zpráv z ovladačů, o zobrazení skóre a o jeho uložení do databáze. Konfigurace aplikace je realizována jednoduchým webovým rozhraním.

Jako hlavní jazyk aplikace byl zvolen Python a to z několika důvodu. Tím hlavním je fakt, že Python umožňuje velmi jednoduchou a expresivní manipulaci s datovými strukturami, což jej dělá výborným jazykem pro tvorbu prototypů. Další důvod je možnost vyvíjet v Pythonu na různých platformách. RPi samo o sobě nemá dostatek výkonu, aby se dali vyvíjet aplikace přímo na něm v rozumném pohodlí. Díky multiplatformní povaze jazyka Python jsem schopen tuto aplikaci vyvíjet a ladit na klasickém osobním počítači s operačním systémem Windows, a následné výsledný kód do RPi přenést a spustit. V neposlední řadě roli ve výběru Pythonu také hraje skutečnost, že Python je velmi všestranný a je možné v něm vyvíjet cokoliv od desktopových aplikací, přes webové služby až po aplikace strojového učení.[6]

Aplikace dělá čtyři věci v jedné aplikaci, a sice servíruje konfigurační web, vytváří GUI pro zobrazení aktuálního skóre, naslouchá informacím z vysílače a navíc ukládá data do databáze.

```
if __name__ == '__main__':
    gui = GUI_THREAD()
    gui.start()

    web = WEB_THREAD()
    web.start()

    controller.listen()
```

Výpis 6: Inicializace aplikace přijímače na RPi

Kód ve výčtu 6 je vstupním bodem do aplikace. V této části kódu se inicializují vlákna, která mají jednotlivé úkoly na starosti a na hlavním vlákně se začne naslouchat datům z ovladače. Nutno podotknout, že aby se kód pro naslouchání přes rádio mohl volat z Pythonu, musí se RPi verze knihovny RCSwitch „zaobalit“. Tento proces zde však probírán nebude.

5.1.1 SessionManager

Než se pustím do popisu jednotlivých součástí přijímače, zmíním se o třídě, která dělá jádro celé aplikace, a sice třídy *SessionManager*. Tato třída totiž drží veškeré informace o právě probíhající hře. Kromě toho se také stará o veškeré manipulace s ní jako přičítání, korekce, resetování i ukládání. Třída také funguje jako jakýsi podavač zpráv. Ten je implementován jako zásobník (*LIFO*). Ihned po přijetí zprávy se zpráva příjemcem zpracuje (v našem případě zobrazí) a zahodí. Stojí za to také podotknout, že jelikož třída drží stav, je nutné volání do této třídy zaobalit zámkem, aby nedocházelo souběhu a stav se tak nestal nepředvídatelným.

5.1.2 GUI

GUI, Graphical User Interface, je v aplikaci realizováno velmi známou knihovnou pro tvorbu GUI, Qt, lépe řečeno jeho Python verzí PyQt. Toto GUI neobsahuje nic jiného, než jednoduchý ukazatel skóre a přezdívky hráčů. Dokáže i konzumovat zprávy *SessionManageru*, jak bylo dříve popsáno, a ty zobrazovat. Ukázku GUI můžeme vidět na obrázku 11.

Obrázek 11: Ukázka GUI aplikace

Nepůjdu zde do detailů, jak můžeme takto jednoduchého GUI pomocí knihovny PyQt dosáhnout, neb to není cílem této práce. Nastíním ale, jak funguje komunikace mezi GUI a třídou SessionManager viz. výpis 7.

```
from SessionManager import session, sessionLock

with sessionLock:
     session.subscribe(self.update)
```

Výpis 7: Zahájení odběru zpráv

Nejprve získám zámek k instanci SessionManageru a poté na ní zavolám metodu *subscribe*, které předám jako argument funkci, která se zavolá, jakmile se změní stav SessionManageru. Metoda *update* je ukázaná ve výpise 8.

```
def update(self, *__args):
      player1name = session.get_player1_name()
      player2name = session.get_player2_name()
      session_in_progress = session.is_in_progress()
      score = session.get_score()
      msgs = session.get_messages()
      self.set_message(msgs)

      self.player_names = ' : '.join([player1name or 'Nobody', player2name or '
          Nobody'])
      self.status_text = 'Session in progress!' if session_in_progress else '
          Session is not in progress'
      self.score_text = ' : '.join([str(n) for n in score])

      return super(GraphicInterface, self).update()
```

Výpis 8: Aktualizace GUI

Metoda nedělá nic jiného, než že nastaví vlastnosti třídy na hodnoty ze SessionManageru a následně zavolá přetíženou metodu na předkovi. Ta zavolá námi přetíženou metodu *draw*, která obsahuje logiku pro vykreslení obsahu.

5.1.3 Web

V této části proberu webové rozhraní aplikace. Toto rozhraní slouží ke konfiguraci, především k přihlášení hráčů do aplikace. Pro tyto účely jsem zvolil framework *Flask*[8]. Tento framework je označován jako *„microframework“*. To znamená, že má velmi málo závislostí a je postaven především na rozšiřitelnosti. To není ale jediný důvod, proč je pro aplikaci ideální. Flask totiž navíc obsahuje i jednoduchý http server, nad kterým Flask může běžet a velmi jednoduše se s ním pracuje (běžně se používá jako vývojářský server). Mimo to Flask implementuje vzor MVC, což znamená, že umožňuje oddělit logiku zpracování dotazu od logiky uživatelského rozhraní v HTML. Nejprve spustíme server podle výpisu 9.

```
def startServer(debug=False):
    initFlask()
    app.run('0.0.0.0', 8080, debug=debug)
```

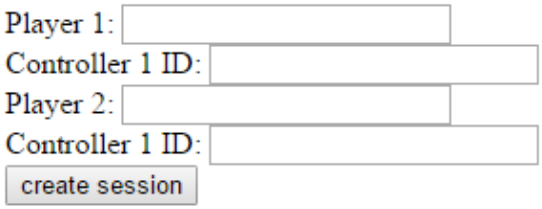
Výpis 9: Inicializace webového serveru pro konfiguraci

Prvním parametrem je adresa. Adresa 0.0.0.0 říká, aby server naslouchal na všech dostupných rozhraních na portu 8080. Poslední argument nastavuje server na *debug* režim. Debug režim přidává extra funkce pro ladění. Například při chybě naservíruje stránku s detailním popisem chyby, včetně výčtu call stacku (což není žádané mimo debug režim). Jelikož není prostor, abych dopodrobna popsal celý Flask, pokusím se jej alespoň nastínit.

```
@app.route("/", methods=['GET', 'POST'])
def index():
    form = PlayersForm(request.form)
    if form.validate_on_submit():
        new_session(form)
    return render_template("index.html", form=form, gameSession=session)
```

Výpis 10: Ukázka frameworku Flask

Ukázku Flask frameworku můžeme vidět ve výčtu 10. Dekorátorem v prvním řádku registruji funkci *index* pod kořenovém URL stránky a říkám, že metoda akceptuje metodu *GET* a *POST*. O veškeré směrování dotazu na definované funkce se stará Flask, stejně tak jako o rozložení dotazu do logických struktur, které jsou v mojí funkci dostupné. V metodě vytvářím instanci třídy *PlayersForm* a vkládám do ní formulářová data z dotazu (request.form jsou data rozložená z POST dotazu). Instance se potom ptám, jestli jsou data validní. Pokud ano, zavolám metodu, která založí v SessionManageru novou hru. Nakonec navrátím odpověď ve formě vykresleného HTML. HTML se vykresluje za pomoci šablon, kterým předávám data, která se do šablony vloží. V tomto případě vykresluje jednoduchý formulář, do kterého uživatel zadá jména hráčů a založí hru. Po založení už stránka pouze podá informaci, že hra probíhá a v takovém případě jí jde zrušit. Rozhodnutí o tom, má-li se zobrazit formulář či tlačítko ke zrušení hry, se děje ve výše zmíněné šabloně.



Obrázek 12: Webové rozhraní

Jak můžeme vidět na obrázku 12, webové rozhraní je velmi jednoduché, ovšem pro potřeby aplikace naprosto dostačující. Kromě jména hráčů vyžaduje formulář také ID ovladačů. Tato pole slouží pro aplikaci přijímače na to, aby věděl, jakým zařízením má naslouchat.

5.1.4 Ovladač

Dosud jsem popsal, jak funguje konfigurace hry a jak se zobrazuje skóre. Touto kapitolou se zaměřím na další klíčový a zároveň poslední prvek aplikace a sice ovládání hry. Na začátku jsem zmínil, že ve vstupním bodě aplikace se inicializují dvě vlákna pro web a GUI, a že hlavní vlákno poté začne naslouchat datům z ovladačů.

```
def listen(self):
      receiver = RCSwitchReceiver()
      receiver.enableReceive(2)
      num = 0
      while True:
            if receiver.available():
                  received_value = receiver.getReceivedValue()
                  if received_value:
                        self.handle_received_val(received_value)
                  receiver.resetAvailable()
```

Výpis 11: Naslouchání vysílače na RPi

Na první pohled se může zdát, že zbytečně zatěžuji procesor nekonečnou smyčkou, avšak volaná metoda *receiver.available* je blokovací funkce, která čeká na přerušení z modulu přijímače. Ideálně bych zavolal metodu handle_received_val, ta by podle přijaté hodnoty řekla třídě SessionManager, co má se svým stavem dělat a program by mohl normálně pokračovat. Nicméně protože komunikace mezi přijímačem a vysílačem není duplexní, nemám způsob, jak potvrzovat přijatou zprávu. Abych měl jistotu, že zprávu opravdu zachytíme, vysílač vysílá zprávu několikrát za sebou a může se také stát, že ji nezachytím jen jednou. Proto musím opakované zprávy filtrovat. Filtr je velmi jednoduchý, pouze kontroluje, kolik času uběhlo od poslední zprávy, a pokud je časový odstup dostatečně dlouhý (více než půl vteřiny) znamená to, že se jedná o novou zprávu. Tuto logiku zapouzdřuje třída *Counter*, ukázaná ve výpise 12.

```
class Counter(object):
    counter = 0
    last_increment = datetime(2000, 1, 1)

    def reset_counter(self):
        self.counter = 0
        self.last_increment = None

    def should_reset_counter(self):
        return self.last_increment and (datetime.now() -self.last_increment).
            total_seconds() > timedelta(seconds=0.5).total_seconds()

    def increment_counter(self):
        self.counter += 1
        self.last_increment = datetime.now()
    # Confirm
    def P1_CLICK(self):
        with sessionLock:
            session.increment_p1_score()
```

Výpis 12: Eliminace duplicitních zpráv z vysílače

Po přičtení skóre SessionManager upozorní GUI na změnu stavu a GUI se překreslí s přičteným stavem.

Nutno podotknout, že logika přijímání a zpracování zprávy byla napsána pro podporu AM prototypu, nicméně může být jednoduše zaměněna za implementaci, která podporuje i další ovladače.

5.2 Úvod do návrhu systému

Zatím jsem ukázal, jak byli vytvořeny jednoduché dálkové ovladače a i přijímač, který zobrazoval na monitor skóre. Problém tohoto systému je, že nesplňuje všechny požadavky. Je to systém uzavřený sám do sebe. Pro komunikaci s ovladači používá svůj vlastní protokol a nebyl navržen tak, aby používal standardní rozhraní pro komunikaci ven do sítě. Odpadá u něj tedy modularita i rozšiřitelnost. Vzhledem k tomu, že konfigurace také probíhá přes jednoduchý *on-premis* web, vytrácí se i intuitivnost a dostupnost. Všechny tyto aspekty mají za následek to, že systém je připraven pouze na jeden přesně definovaný use-case.

Podívejme se teď ovšem na náš problém z opačné strany. Co kdybychom byli schopni systém navrhnout tak, že místo toho, abychom stavěli systém na jednom use-case, systém bude připraven na rozšíření pro libovolný use-case? Bude dostatečně modulární na to, aby se jednoduše daly zaměňovat jednotlivé jeho části, bude komunikovat přes jasně definované, standardní rozhraní tak, aby byl rozšiřitelný, a navíc bude splňovat veškeré další požadavky. Návrh a implementaci přesně takového systému se pokusím v následujících kapitolách přiblížit.

Pro vývoj jsem se rozhodl použít technologii *Microsoft ASP .NET*[9] nad kterým budu vyvíjet jazykem C#. Důvodů pro takovou volbu je celá řada. Pokusím se je nyní heslovitě nastínit.

- **C#:** Velmi vyspělý multiparadigmatický programovací jazyk, který umožňuje vysokou míru flexibility.
- **ASP .NET:** Drasticky snižuje množství kódu pro psaní rozsáhlých webových aplikací. Všechny jeho části jsou modulární a konfigurovatelné.[9]
- **ASP .NET Identity:** Framework pro rychlou implementaci správy uživatelů a jejich rolí.[10]
- **Microsoft Azure:** Cloudová platforma vyvíjená Microsoftem. Mezi její výhody patří snadné a rychlé nasazení aplikace, škálovatelnost, intuitivní a jednoduchá zpráva přes webový portál atp.[13]
- **Microsoft support:** .NET Framework i všechny jeho součásti jsou neustále udržovány a vyvíjeny firmou Microsoft.
- **Vývojové nástroje:** Visual Studio je považováno jako jedno z nejlepších vývojových nástrojů vůbec.

5.3 Použití

Na jedné straně máme uživatele. Uživatel má k sobě přiřazenou kolekci ukazatelů skóre a ovladačů. Můžeme si uživatele představit jako jednotlivce, který má svůj ovladač a na jeho telefonu mu běží aplikace ukazatelé skóre, jakou jsem ukázal v kapitole 4.1 a chce hrát soutěž kdekoliv, kde je dostupná síť, nebo pracovníka sportoviště, který má na starosti několik tenisových kurtů v hale a pro každý kurt jeden monitor s RPi na kterém běží ukazatel skóre a k nim sadu ovladačů.

Každá hra začne tak, že jednotlivec či pracovník, na webovém portálu založí tzv. sezení. Při tvorbě tohoto sezení zadá identifikátor ovladačů a k nim ukazatelé skóre, které chce spárovat s hrou a ovladači.

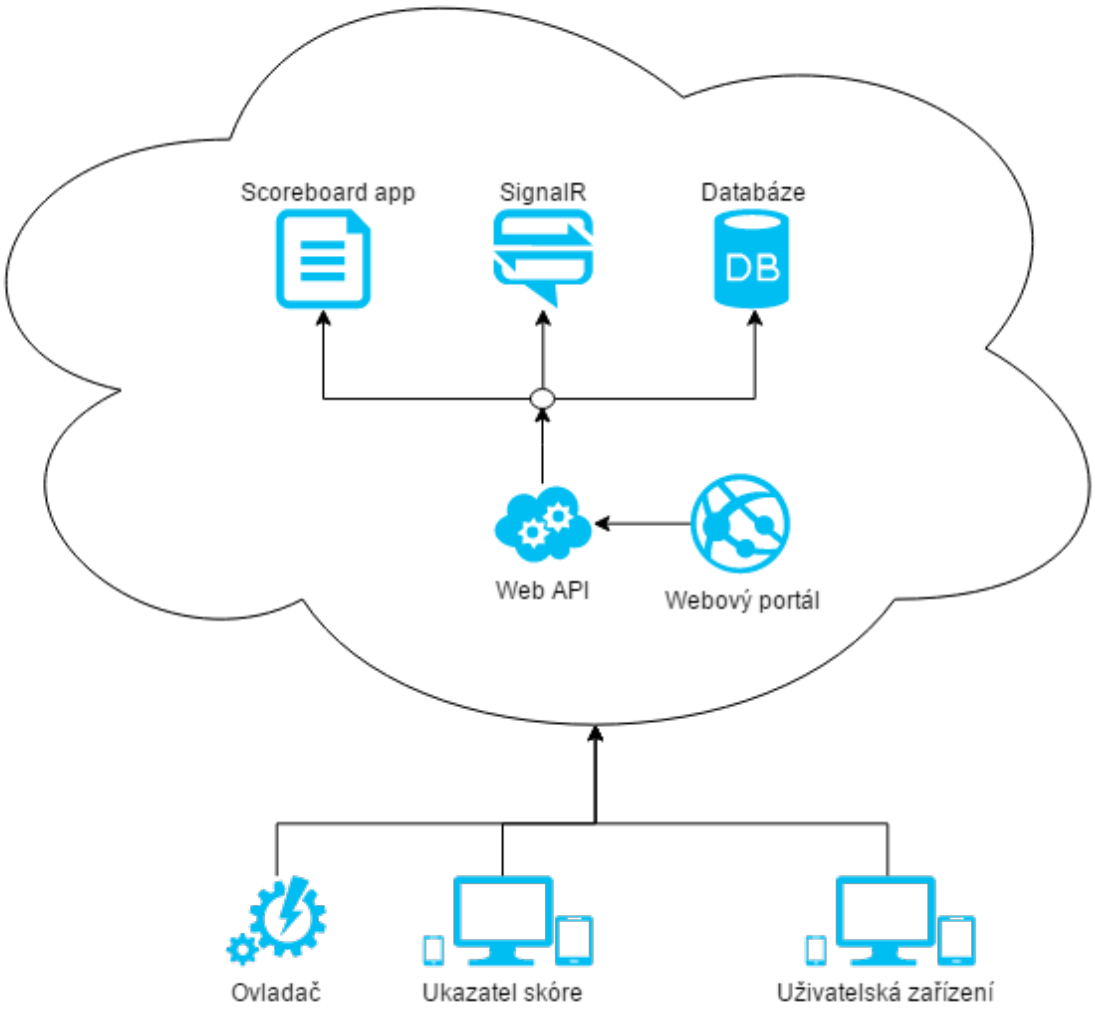
Po vytvoření sezení už probíhá hra a hráči si na svých ovladačích přičítají skóre, které se jim zobrazuje v aplikaci ukazatele skóre.

Po skončení hry hráči stisknou gesto pro jeho uložení. Sezení se tak označí za dokončené a v tu chvíli se stává neměnné.

Pokud hráč, který zakládal sezení zadal i identifikátor druhého hráče, oběma se tato hra zařadí do statistik, které pak mohou na webovém portálu zobrazit. Stejně tak si může počínat i pracovník ve sportovišti. Může přihlásit hráče do hry, následně spárovat jejich ovladače s ukazatelem. Nebyl by ani problém, kdybychom řekli, že ovladače nechceme hráčům prodávat, ale místo toho je zapůjčovat na sportovišti.

5.4 Architektura

V první řadě je třeba říci, čeho se v návrhu snažím dosáhnout. Budu se především snažit o co nejlepší odezvu, modularitu a rozšiřitelnost. Později v této kapitole vysvětlím, jak takových požadavků dostojím. Na obrázku 13 můžeme vidět architekturu celého systému.



Obrázek 13: Architektura systému

5.5 Použité návrhové vzory

5.5.1 Dependency injection

Dependency injection je návrhový vzor, který spočívá v automatickém vyřešení závislostí mezi jednotlivými komponenty programu. Jednotlivé typy se nejprve zaregistrují pod svým rozhraním do takzvaného IoC (Inversion Of Control) kontejneru. Kontejneru je třeba říct, jaký má mít typ životní cyklus[11]. Např. má se při každém požadavku na typ vytvořit nová instance anebo má být instance držena po celý běh programu?

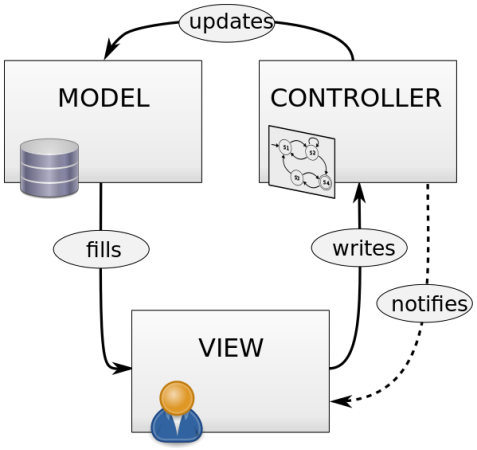
Kontejnery mohou fungovat ve 2 režimech a sice v reflexivním režimu, kdy se při vytváření typu kontejner pokusí předat všechny závislosti, které jsou definovány v jeho konstruktoru, nebo v režimu „on demand“, tedy režim, při kterém se závislost řeší až ve chvíli, kdy je potřeba a to tak, že se na typ dotazuje přímo kontejneru.

Dependency injection je také velmi důležitý z pohledu testovatelnosti. Při psaní unit testů můžu totiž závislosti typů nahrazovat falešnými typy, které implementují stejné rozhraní. Toto umožňuje testovat bez vedlejších efektů a podstatně rychleji izolované části kódu.

Je důležité také zmínit, že veškerou implementaci závislostí schovávám za rozhraní a jejich implementace můžu libovolně měnit. To dělá aplikaci modulární.

5.5.2 MVC

MVC neboli Model View Controller je vzor, který od sebe odděluje řídící logiku, datový model a uživatelské rozhraní[11]. Prakticky to znamená, že se tyto části dají vyvíjet samostatně a jejich modifikace nemají dopad na další části. V naší aplikaci však je lepší pojem uživatelské rozhraní zaměnit za datový pohled, protože v naší aplikaci nevracíme rozhraní, ale datovou strukturu.



Obrázek 14: Vzor MVC (www.wikipedia.org)

5.6 Subsystémy

5.6.1 Ukazatel skóre

V kapitole 5.1 jsem představil jednoduchý ukazatel skóre, který ale nesplňoval všechny požadavky. Představte si například situaci, kdy bych chtěl rozšířit funkcionalitu tak, aby všichni, kteří mají zájem, dostávali notifikaci do svých mobilů o tom, že právě vyhrávám hru. Co když třeba chci, aby hráči mohli ukazatele skóre používat třeba na své domácí (chytré) televizi či mobilním zařízení, přitom ale zachovat stejnou funkcionalitu? Mohl bych mít pro všechny platformy aplikaci klienta, který se baví se systémem přes nějaké rozhraní, ale udržovat kód na tolika platformách je velmi nešikovné.

Můžu s jistotou říct, že každá platforma umožňující připojení k internetu, jako jsou chytré televizory, mobily, tablety atp., umí zobrazovat webové stránky. Proto řešením, jak využívat funkcionalitu ukazatele skóre, je jednoduchá webová aplikace. Kromě výhod spojených s multiplatformností existují ještě další výhody např. aktualizace aplikace. Kód aplikace pouze vystavím na internet a na klientech je pouze to, aby zdrojový kód načetli a stránku zobrazily.

Webová aplikace dnes zdaleka nemusí běžet jen v okně prohlížeče. Její kód například můžeme zaobalit do *Electronu*, nástroje, který z webové aplikace dokáže udělat klasickou desktopovou aplikaci. Další možností je například použít framework *Cordova*, který jí zaobalí do nativní Android aplikace.

5.6.2 SignalR

Tradiční aplikace „tenkého“ klienta používají pro výměnu dat se serverem bezestavový protokol HTTP a na data se doptávají „on demand“, tedy až v momentě, kdy jsou potřebné. Jinými slovy, většinou se doptává klient na data, jen v málo případech dokáže server sám posílat data na klienta. Posílání zpráv ze serveru na klienta bez dotazu se označuje jako „push“.

Způsobů, jak tento „push“ implementovat existuje více. Příkladem může být tzv. Long Polling. Jde o metodu, kdy http spojení sice není perzistentní, nicméně klient se o nová data dotazuje v pravidelných intervalech. Tyto a další metody se používají, neimplementuje-li klient *Websocket*. *Websocket* je protokol, který umožňuje plně duplexní komunikaci mezi klientem a serverem. SignalR[12] je součást ASP .NET, která zastřešuje použití těchto metod. Jeho výhoda je právě v tom, že dokáže určit jaké jsou schopnosti klienta a na základě toho domluvit se serverem příslušnou metodu pro komunikaci. Mě SignalR poslouží jako prostředek k tomu, abych ukazateli skóre dal vědět, že se skóre změnilo.

Ještě doplním, že pro SignalR také existuje množství klientských knihoven pro různé platformy včetně Javascriptu i ve formě JQuery pluginu.

5.6.3 Webový portál

Další součástí systému bude webový portál, který bude mít na starost zobrazování dat, vytvoření sezení a autentizaci uživatele. Uživatel systému bude mít na portálu vlastní účet a bude schopen zobrazovat si statistiky svých soutěží v přehledných a intuitivních grafech. Dále také bude schopen párování svých ovladačů se svými ukazateli skóre. Tyto povinnosti však nejsou definitivní a portál se musí být schopen dále rozšiřovat o další rysy.

Klasické dynamické stránky mají určitě nevýhody. Velké množství HTML, CSS a JS kódu produkuje velmi těžko udržitelný kód. Řešení je použít framework, kterým bych byl schopen rozdělit jednotlivé části HTML kódu do menších částí, do zaměnitelných, lehce udržovatelných komponent. Navíc bych chtěl splnit požadavky na odezvu a intuitivnost. Abych pokaždé nemusel načítat celý kód HTML stránky, použiji formát JSON pro předávání dat mezi klientem a serverem přes tzv. *Asynchronous JavaScript and XML* neboli *AJAX*. To ale znamená, že komponenty musí obsahovat i množství logiky v Javascriptu, které bych chtěl oddělit od designu tak, abychom vnitřní logiku a prezentační vrstvu mohl udržovat zvlášť.

Dalším problémem je samotný Javascript. Za poslední roky se Javascript velmi rozšířil ve webových stránkách. Tento jazyk má ale vlastnosti, které jej činí nevhodný pro vývoj větších aplikací. Příkladem může být fakt, že Javascript je dynamicky typovaný a jako takový odhaluje chyby způsobené špatnou typovostí objektů až za běhu, což může znamenat relativně velké zpomalení vývojového procesu, nemluvě o nekonzistenci typů a s tím spojeným špatně udržovatelným kódem. Řešením je použít super set Javascriptu, který by dovoloval kompilaci s kontrolou typů ale i další rysy pro lepší udržitelnost kódu.

Z výše uvedených důvodů jsem se rozhodl použít framework Angular[14]. Angular je open-source framework vyvíjený v Googlu pro vývoj SPA. Zjednodušuje vývoj webových aplikací i jejich testování tím, že poskytuje implementaci MVC architektury. Angular se vyvíjí v jazyce Typescript vyvíjeným Microsoftem, který umožňuje kompilaci do Javascriptu se statickou kontrolou typů.

Protože budu chtít vykreslovat data do grafů, zvolil jsem pro tyto účely knihovnu Highcharts[15], kvůli jednoduchosti a designu.

5.6.4 Databáze

Dalším subsystémem je databáze. Místo, kde budeme ukládat nejen skóre ale i uživatele, jejich ovladače a ukazatelé skóre.

Navzdory stále rostoucí popularitě *NoSQL* persistentních vrstev jsem se rozhodl použít *SQL Server* od firmy Microsoft. Mezi důvody proč použít SQL databázi jako hlavní datové uložiště patří například jednoduchost struktur, ve kterých jsou data uloženy, parametrizované dohledávání dat, škálovatelnost či normalizace, která zamezuje tvorbu duplicit.

I přes tyto argumenty však mají NoSQL databáze v takovémto systému své místo. Můžeme díky nim především zrychlit odezvu systému. NoSQL databázi totiž můžeme předřadit před samotnou SQL databázi a jednotlivé datové pohledy z SQL v ní ukládat pro rychlejší přístup. V tu chvíli tedy NoSQL databáze slouží jako *cache*. Problémem je, jak takovouto *cache* následně zbavit platnosti, když se změní data. Řešením je oddělením přístupu do databáze na čtení a na zápis. Zapisovací část jednoduše upozorní čtecí část, že se data změnila. Tomuto vzoru se říká *Command And Query Responsibility Segregation* neboli CQRS. Toto však v našem systému momentálně implementováno nebude.

5.6.5 Entity Framework

Již jsem zmínil, že jako hlavní úložiště dat budeme používat SQL Server. Pravda je ale taková, že na samostatné databázové technologii, při pohledu z vnějších vrstev, nezáleží. Dobrou praktikou je oddělit databázovou vrstvu od dalších vrstev tak, aby bylo možné databázovou vrstvu zaměnit bez narušení dalších vrstev.

Entity Framework[11] je technologie pro přístup k datům, vyvíjená a udržovaná firmou Microsoft. EF je objektově-relační *mapper*, který umožňuje práci s relačními daty za použití doménově-specifických objektů. Eliminuje míru kódu, který se pro přístup k datům musí napsat. EF navíc funguje nad množstvím databázových technologiích.

Existují 2 způsoby práce s EF. Tím prvním je tzv. *Database-First Approach*. To znamená, že se nejdříve založí databáze, definuje se v ní relační model a následně se řekne EF aby z tohoto modelu vygeneroval třídy, reprezentující jednotlivé entity v databázi. Druhým způsobem je *Code-First Approach*. Toto je opačný způsob, kdy se založí třídy reprezentující jednotlivé entity, ze kterých EF následně vyrobí relační model, případně vytvoří databázi.

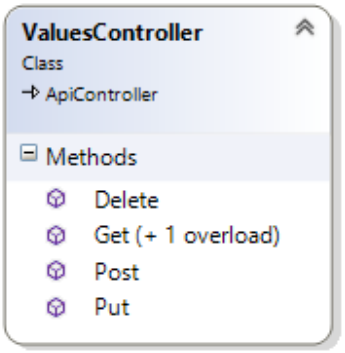
5.6.6 Web API

V předchozích kapitolách jsem stručně představil technologické součásti našeho systému. Všechny součásti je ale třeba něčím spojit.

Jako řešení jsem se vydal cestou *RESTful* služby. REST[17], neboli Representational state transfer, je standard pro rozhraní služby (v našem případě http služby), který definuje uniformní sadu beze-stavových operací pro manipulaci s obsahem. Nejběžnější formáty pro komunikaci s RESTful službou jsou *JSON* a *XML*. REST cílí především k rychlosti, spolehlivosti a škálovatelnosti. Důležité je také zmínit, že takovou službu může využívat kterékoliv zařízení schopné komunikovat přes http. ASP.NET Web API[11] je framework pro tvorbu takových služeb.

Web API používá MVC architekturu. Vše se tedy točí kolem tzv. kontrolérů. Při implementaci těchto kontrolérů je třeba zachovat jmenné konvence, aby je framework dokázal správně použít. Pokusím se přiblížit, jak ve Web API vytváří sada jednoduchých metod pro manipulaci s obsahem.

Na obrázku 15 vidíme diagram třídy *ValuesController*. Je důležité, aby jméno třídy končilo názvem Controller a třída dědila z třídy ApiController. Takto definovaná třída stačí, aby framework byl schopen rozložit http request, jeho parametry rozmístit do vnitřních struktur a podle URL předat request našemu kontroléru. Definice metod je také velmi důležitá. Podle *http metody* requestu framework předá request konkrétní metodě. Názvy metod tedy musí začínat názvem *http metody*. Toto chování se dá ovlivnit anotacemi nad metodami. Další faktor na základě kterého se framework rozhoduje, kterou metodu zavolá, jsou její parametry. Pokud request neobsahuje žádné parametry v těle ani v URL, zavolá se metoda bez parametrů. Pokud ano, zavolá se metoda přijímající tento parametr.



Obrázek 15: Ukázka kontroléru

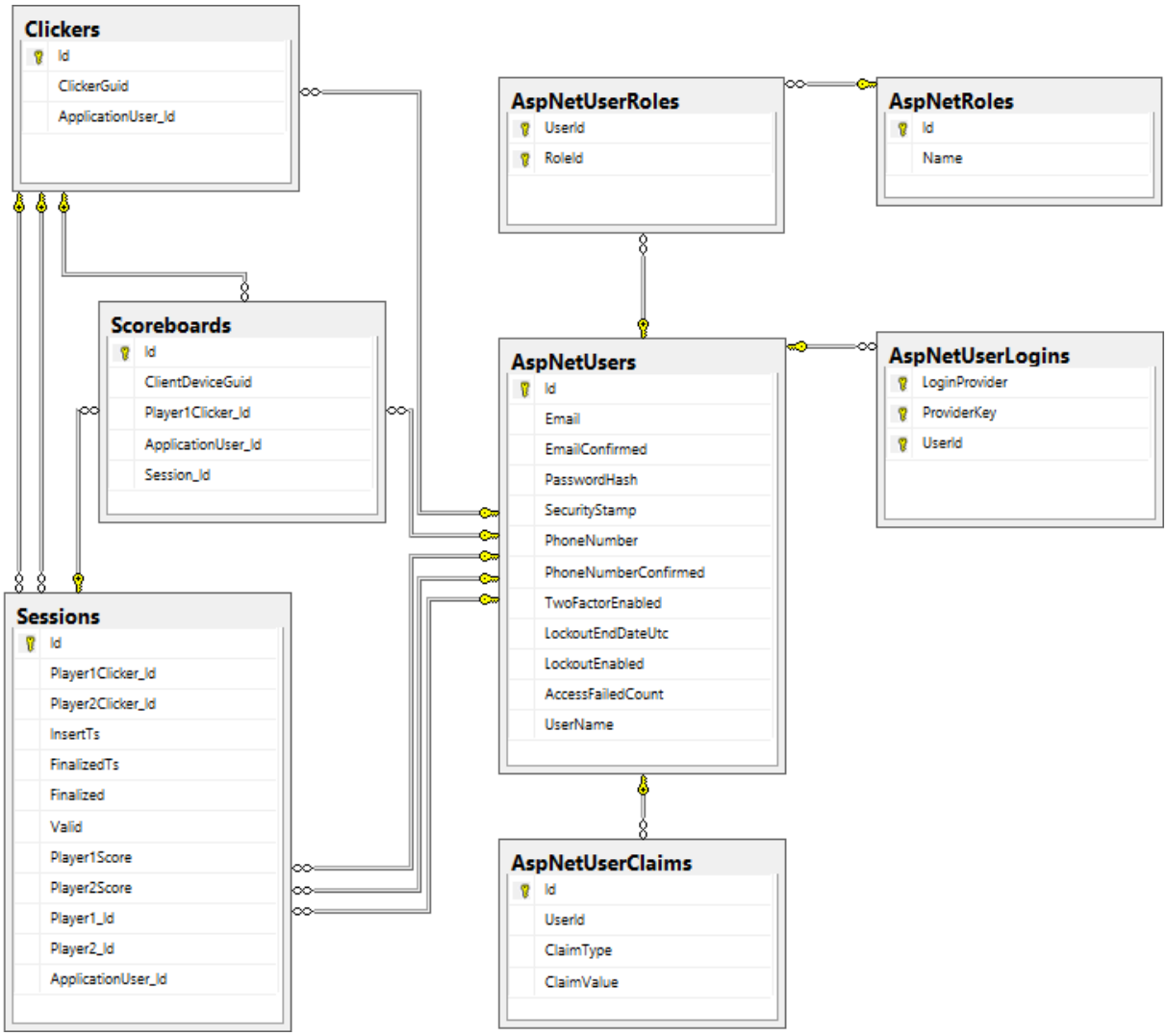
Součástí Web API je také implementace autorizačních a autentizačních mechanismů, které jsou součástí ASP NET Identity, které také využijeme.

6 Implementace systému

6.1 Databázový model

V této kapitole představím ER diagram databáze, kterou aplikace používá a zároveň ukážu účel jednotlivých entit, se kterými budu pracovat. Diagram můžeme vidět na obrázku 16.

Tabulky, které mají v názvu „AspNet“ jsou vygenerovány ASP .NET frameworkem, konkrétně se jedná o *ASP .NET Identity*, framework pro správu autorizace a autentifikace uživatele.



Obrázek 16: Databázový model

Následuje popis jednotlivých entit.

- **Clickers:** Tato entita slouží pro evidenci jednotlivých ovladačů, popsaných v dřívějších kapitolách. Existuje především k autentizaci jednotlivých ovladačů a jejich přiřazení k ukazatelům skóre, aby systém věděl, které skóre eventuálně přičíst.
- **Scoreboards:** Tabulka zařízení nebo aplikací s ukazateli skóre. Slouží pro identifikaci ukazatelů.
- **Session:** Neboli sezení. Jedná se o spojovací (párovací) tabulku mezi ovladači a ukazateli skóre. Tato entita se vytváří při každé hře a kromě informací o hře obsahuje také informaci o tom, který ukazatel skóre je ovládaný jakým olvadačem.
- **AspNetUsers:** Tabulka uživatelů se všemi příslušnými náležitostmi pro bezpečnou autentifikaci a autorizaci. ASP .NET Identity umožňuje mimo to i správu rolí pro jednotlivé uživatele. Tu však v našem systému zatím používat nebudeme.

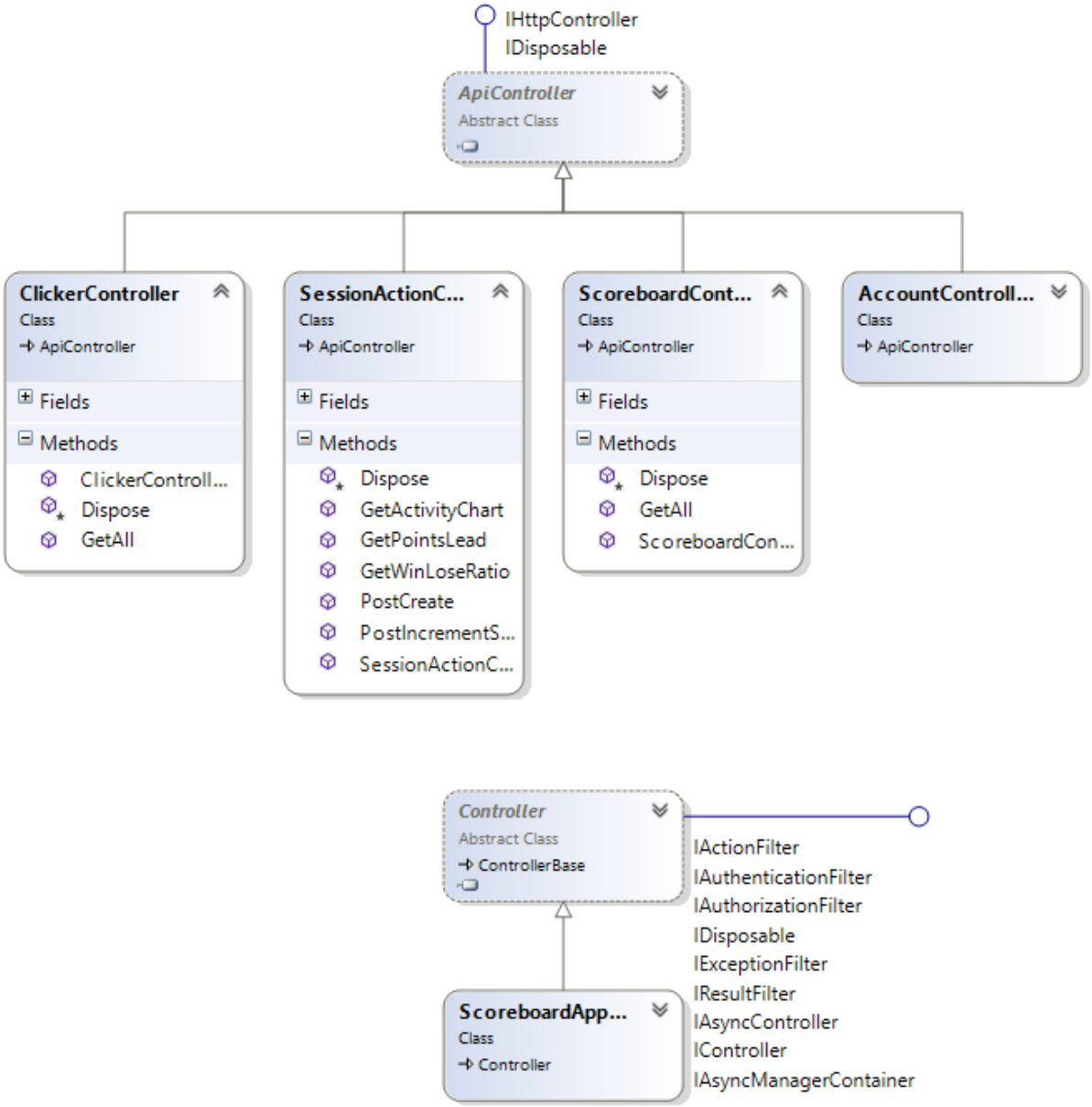
6.2 Cloudová aplikace

Aplikaci sestavím především s pomocí modelů, MVC kontrolérů a služeb. Model je pouze paměťovou reprezentací databázových entit, které jsem popsal dříve. MVC kontroléry obsahují vykonávací logiku tak, jak je tomu v klasickém MVC vzoru. Služby obsahují dodatečný kód, který je z hlediska dobrých praktik nevhodné dávat kamkoliv jinam.

Různé služby mají různé závislosti nebo mají různý životní cyklus. Je krajně nevhodné starat se o vytváření jejich instancí ručně. Navíc by na to bylo třeba vědět o jejich implementaci, a tak schovávání implementace za rozhraní ztrácí smysl. Dělá to kód méně znovupoužitelný a také špatně testovatelný. Pro správu závislostí tedy používáme návrhový vzor Dependency injection.

6.2.1 Kontroléry

Jak jsem dříve zmínil, celá aplikace se zakládá na ASP MVC, tedy frameworku využívající vzor MVC. Pokusím se v této kapitole přiblížit kontroléry používané v aplikaci a popsat jejich jednotlivé metody. Diagram kontrolérů můžeme vidět na obrázku 17.



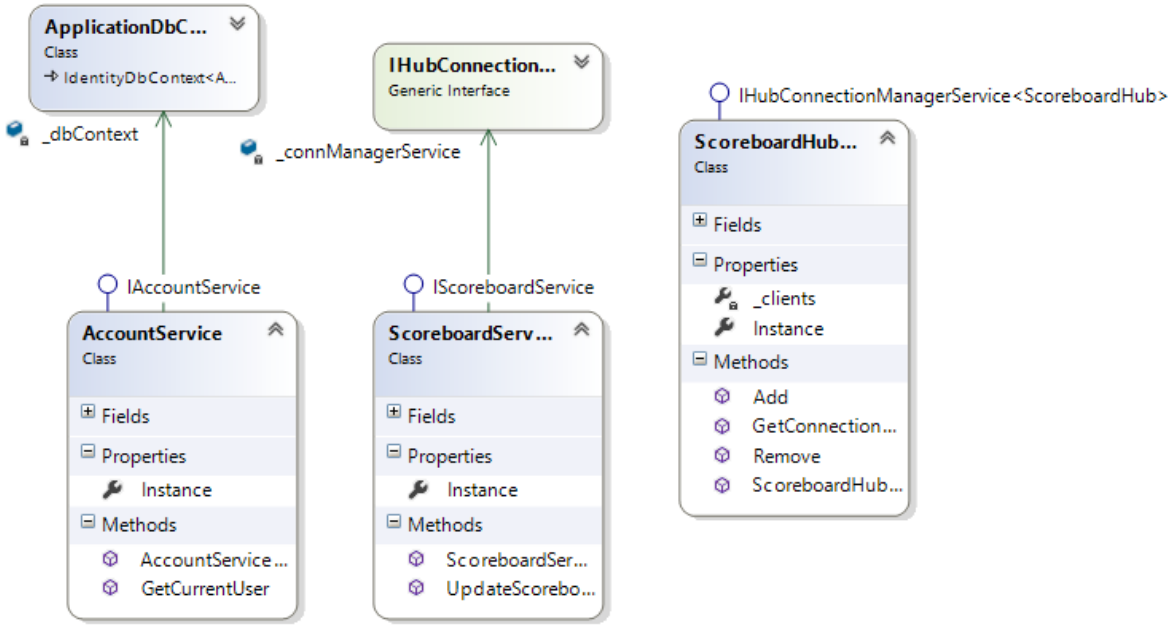
Obrázek 17: Třídní diagram kontrolérů

Všechny kontroléry se v zásadě dají spojit s jednotlivými entitami v databázovém modelu. Tato vlastnost je cílená, aby jeden kontrolér měl na starosti metody pro manipulaci s jednou entitou. Tyto kontroléry se navíc mapují na URL, které odpovídá entitě. Například pro AccountController bude URL `http://domena.cz/api/account/"název_metody"`. Jak jsem řekl v minulých kapitolách, rozhraní není plně RESTful, jelikož není uniformní napříč entitám, nicméně další výhody RESTful jsou stále zachovány.

- **ClickerController:** Kontrolér pro ovladače. Obsahuje pouze metodu GetAll, která vrátí seznam všech ovladačů.
- **ScoreboardController:** Kontrolér pro ukazatele skóre.
- **SessionActionController:** Kontrolér, který obsahuje metody pro manipulaci se sezením. Toto zahrnuje párování ovladačů a ukazatelů, přičítání skóre, ukládání atd. Kromě toho také obsahuje metody pro získání dat pro zobrazování různých statistik. Např. GetActivityChart vrátí kolekci týdnů a počet her, které v nich byly uskutečněny.
- **AccountController:** Toto je záležitost ASP .NET Identity. Obsahuje metody pro autentizaci a autorizaci uživatele.
- **ScoreboardAppController:** Tento kontrolér na rozdíl od ostatních nedědí z ApiController. Tento totiž nevrací JSON, ale celý kód aplikace ukazatele skóre, do které ale vloží identifikátor zařízení, které si o ní požádalo. SessionActionController obsahuje metody, které připraví data pro graf. Toto chování je značně spojené s rozšiřitelností systému. Chceme-li připravit nějakým způsobem data, abychom je poté v aplikaci klienta zobrazili, přidáme do kontroléru další metodu. Jistě bychom jen mohli vracet seznam všech sezení a z nich následně na klientovy zformulovat data do požadovaného formátu. Toto je ovšem velmi špatné z hlediska rozdělení zodpovědnosti. Klient by měl v ideálním případě data pouze zobrazovat a samotná data by měl připravovat server. Toto dělá kód přehledný a udržovatelný.

6.2.2 Služby

V této kapitole nastíním sadu služeb použitých v aplikaci. Jejich diagram je na obrázku 18.



Obrázek 18: Diagram služeb

- **AccountService:** Jednoduchá služba, které nedělá nic jiného, než že z kontextu požadavku klienta dostává identitu přihlášeného uživatele a podle této identity dohledá jeho záznam v databázi.
- **ScoreboardService:** Tato služba tvoří jádro podsystému pro notifikaci ukazatelů o novém počtu bodů. Obsahuje pouze jednu metodu a sice *UpdateScoreboard*. Tato metoda přijímá jako parametr instanci entity sezení a následně podle identity uvedené v sezení upozorní daný ukazatel.
- **ScoreboardHubManagerService:** Služba, jejíž starost je mapování identit ukazatelů skóre na jednotlivá SignalR připojení. Tato služba umožňuje upozornit konkrétního klienta.
- **ApplicationDbContext:** Umožňuje přístup k datům v databázi přes EF.

6.2.3 Autentizace a Autorizace

V aplikace je třeba autentizovat nejen uživatele ale i jednotlivá zařízení. Musím totiž být schopnen zařízení mezi sebou párovat.

ASP .NET Identity[10] nabízí robustní framework nejen pro autentizaci uživatele, ale také pro správu jeho rolí. Dokonce je velmi jednoduché zamezit jednotlivým rolím přístup k rozhraní. Co se týče našich kontrolérů, stačí atribut *Authorize*, kterému v parametrech vyjmenujeme role, kterým přístup povolujeme.

Pro autentizaci použijeme standardní protokol *OAuth 2*. Díky faktu, že tento protokol používají rozšířené služby, jako například Twitter, Facebook či Google, je dokonce možné získat identitu uživatele z jejich účtu na zmíněných službách. Tento rys však zde implementován nebude.

Krátce popíši proces autentizace. Uživatel nejprve zadá své uživatelské jméno a heslo. Tyto údaje se ověří proti databázi. Následně se pro uživatele vygeneruje token. Tento token se uchovává v paměti serveru mimo kontext požadavku, to znamená, že se udrží napříč požadavky. Token se předá klientovi, ten si jej uchová (cookies, session storage). Pro další komunikaci se tento token s každým požadavkem posílá na server. Na serveru se pak mapují identity na tento token.

Kromě uživatelů je třeba autentizovat i zařízení. Pro tyto účely má každé zařízení unikátní ID v podobě struktury *Global Unique Identifier* (GUID). Tento přidělený identifikátor pošle serveru místo uživatelského jména a následně se validuje proti databázi. Toto neplatí jen pro ukazatelé skóre ale i pro ovladače. Prakticky to funguje tak, že ovladač zadá dotaz na přičtení skóre, který sebou nese i zmíněný identifikátor. Na druhé straně ukazatel skóre když si zažádá kód aplikace, musí s požadavkem zahrnout i svůj identifikátor, který se následně vloží do aplikace a tento identifikátor se pak nese s každým dotazem. Příklad autorizace jde vidět na výpisu 13.

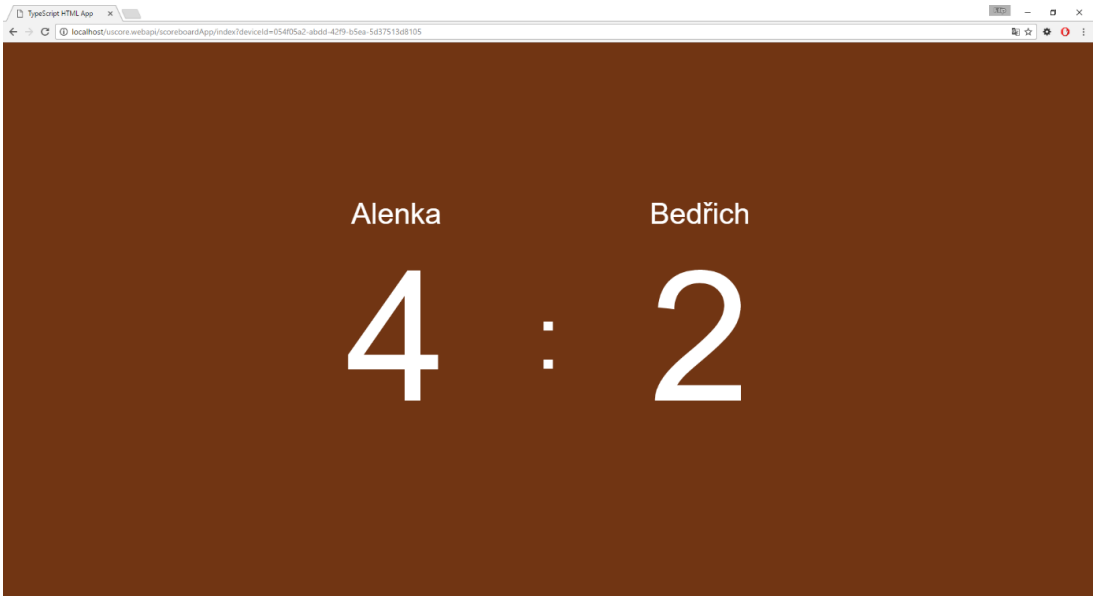
```
private login(): any {
    var loginData = {
        grant_type: 'password',
        username: Globals.deviceGuid, // vistknute id zarizeni
        password: ''
    };

    return $.ajax({
        type: 'POST',
        url: '/Token',
        data: loginData
    });
}

  this.login().done((data) => {
  var at = data['access_token'];

  sessionStorage.setItem('accessToken', at);
  $.signalR.ajaxDefaults.headers = { Authorization: "Bearer " + at };

  this.initHubConnection();
  }).fail((data) => {
       alert("Authorization failed.");
  });
```

Výpis 13: Ukázka autentizace ukazatele skóre

6.3 Ukazatel skóre

V návrhu jsem uvedl, že ukazatel skóre bude implementován jako webová aplikace kvůli podpory různých zařízení, obecné rozšiřitelnosti, jednoduchosti a dalšími výhodami, které sebou tento přístup nese.

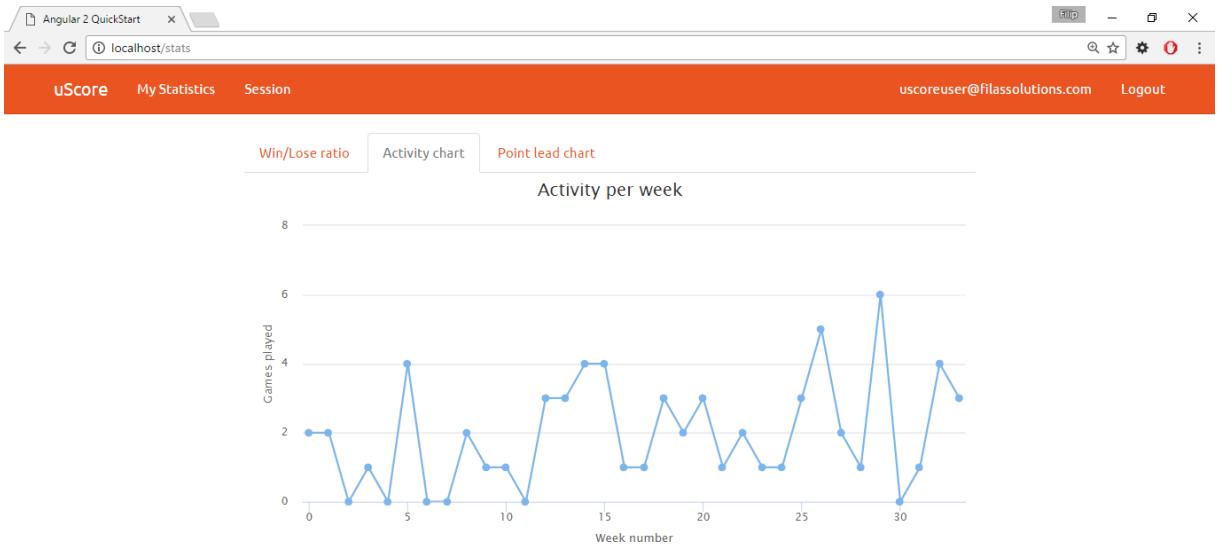
Tato aplikace je napsána za pomoci *HTML5*, *CSS3* a *JQuery*.JQuery[18] je framework vytvářející uniformní rozhraní Javascriptu napříč všemi prohlížeči. Díky principům responzivního designu je aplikace schopná přizpůsobit se jakkoliv velkému displeji. Tyto jazyky jsou navíc mnohem rozšířenější a všestranější než PyQt, který jsem na aplikaci původně používal. Na obrázku 19 můžeme vidět jak nový ukazatel skóre vypadá.



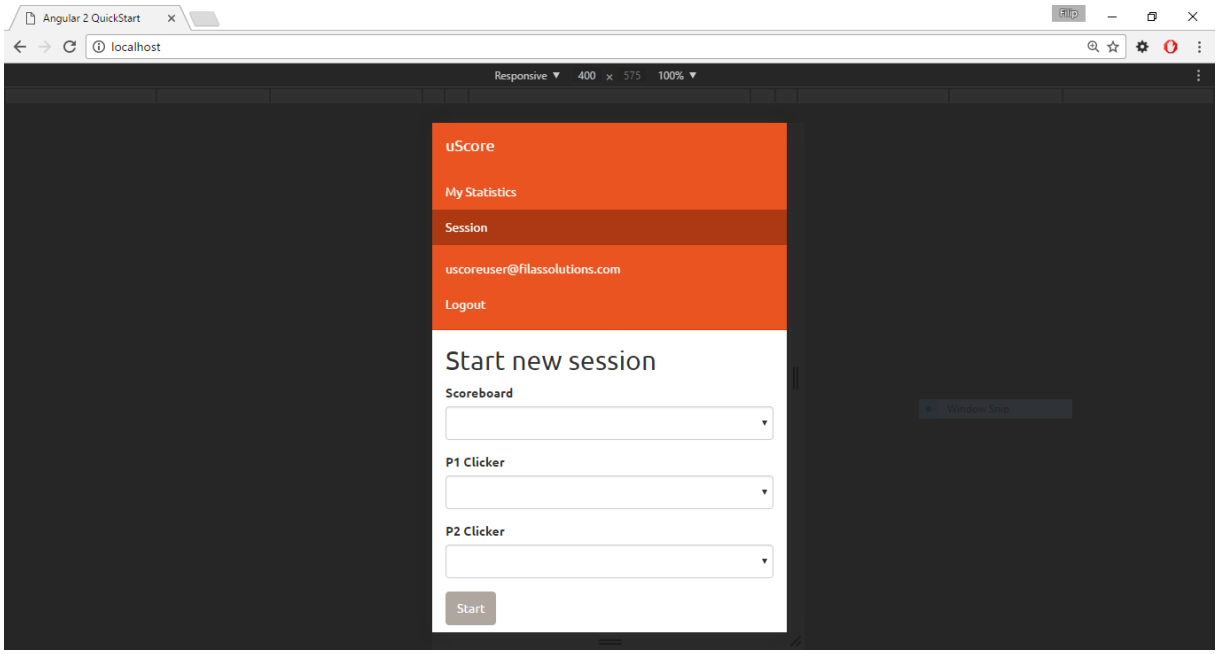
Obrázek 19: Webová aplikace ukazatele skóre

6.4 Webový portál

Tato sekce obsahuje ukázky webového portálu.



Obrázek 20: Ukázka webového protálu - graf aktivity



Obrázek 21: Ukázka webového portálu - responzivní design

7 Měření odezvy systému

7.1 Pomůcky

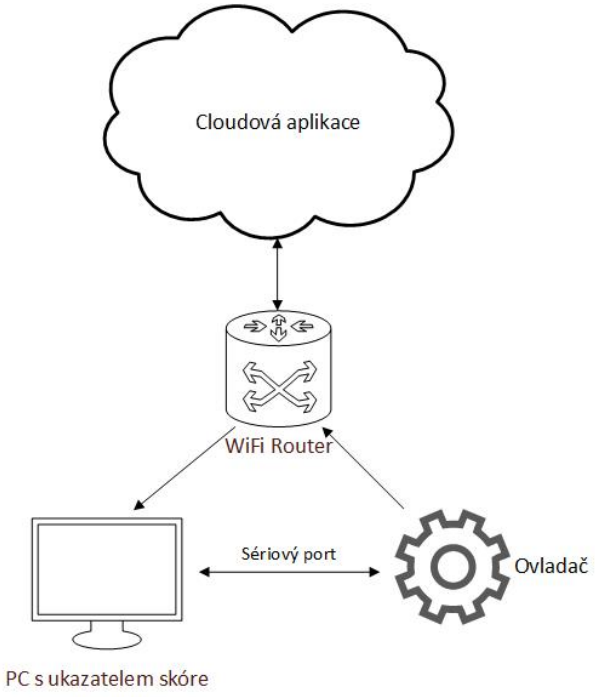
NodeMCU s programem ovladače, aplikace ukazatele skóre, cloudová aplikace, služba Microsoft Azure.

7.2 Cíl měření

Cílem měření je zjistit odezvu systému při přičtení skóre, tedy dobu od požadavku ovladače do cloudové aplikace o přičtení skóre až po zachycení této události ukazatelem skóre.

7.3 Postup měření

Mám ovladač připojený k počítači přes sériový port. Jeho program modifikuji tak, aby před každým odesláním požadavku o přičtení skóre přes sériovou linku poslal zprávu. Na počítači zaznamenávám každou takovou zprávu s časovou známkou. Na počítači také bude spuštěna aplikace ukazatele skóre. Její kód rovněž upravíme, aby při zachycení zprávy o změně skóre zaznamenala časovou známku této události. Obě časové známky od sebe odečtu, čímž získáme časový rozdíl mezi událostmi. Událost přičtení skóre vyvolám celkem 30 krát. Z naměřených časových rozdílů následně získám jejich průměr. Cluodová aplikace je v případě měření hostovaná službou *Microsoft Azure*.



Obrázek 22: Diagram zapojení

7.4 Naměřené hodnoty

č. pokusu	odezva (ms)
1	148
2	123
3	121
4	129
5	109
6	130
7	130
8	118
9	157
10	118
11	132
12	110
13	128
14	118
15	107
16	131
17	116
18	120
19	106
20	129
21	121
22	109
23	117
24	104
25	133
26	117
27	119
28	115
29	126
30	118
31	110
Průměr	121

Tabulka 5: Tabulka naměřené odezvy systému

7.5 Závěr

Průměrná odezva systému činí 121 ms s maximální odchylkou 36 ms, tedy 29,75 %.

8 Závěr

V druhé kapitole jsem ukázal, jak se dá realizovat jednoduchý prototyp dálkového ovládání. Jaké jsou dostupné moduly, jak se používají a jaké jsou jejich výhody a nevýhody.

Ukázal jsem, že byť má AM Rádio modul na 433 MHz dosah větší než 80 metrů, jeho spolehlivost přenesení dat na takovou vzdálenost je menší než 20 %. Na druhou stranu vydrží na tužkovou baterii více než 60 hodin vysílání. Jeho použití tedy vidím spíše v aplikacích s kontinuálním přenosem dat, kde není prioritou spolehlivost přenosu.

Prototyp s HC-05 má 100% spolehlivost pouze na 10 metrů. Poté jeho spolehlivost klesá a při 15 metrech už činí pouze 50 %. HC-05 má však tak nízký odběr, aby vydržel 40 hodin při kontinuálním vysílání na tužkovou baterii.

Poslední prototyp (a také zvolený prototyp pro systém) je NodeMCU. Tato vývojová deska, jejíž srdce tvoří modul ESP8266, má dosah větší než 100 metrů. Při 100 metrech byla naměřena spolehlivost 100 %. Bohužel však má nízkou výdrž. Pouze 20 hodin kontinuálního vysílání na tužkovou baterii. Ovšem pro mé požadavky je to dostačující výdrž, jelikož interval, ve kterém bude vysílat, je v řádech minut.

V dalších kapitolách jsem přiblížil návrh a implementaci cloudové aplikace, která je jádrem celého systému. Popsal jsem jednotlivé komponenty, použité technologie, návrhové vzory i implementaci. Dále popíši, jak přesně jsem splnil veškeré požadavky.

Spolehlivost byla splněna především výběrem spolehlivého modulu pro použití pro dálkové ovládání. Spolehlivost cloudové aplikace je z velké části dána spolehlivostí infrastruktury resp. služby, na které je vystavena v síti Internet. Tou se v této práci nezabývám, protože toto je zodpovědnost platformy, na které je systém hostován (Microsoft Azure). Důležitým faktorem pro spolehlivost cloudové aplikace je ale také funkční kód, ošetřující výjimky, které mohou nastat. Díky použití vzorů Dependency injection a MVC je náš kód dobře testovatelný a tedy méně náchylný k chybám.

Průměrná naměřená hodnota odezvy ovladače a ukazatele skóre byla naměřena 121 ms s maximální odchylkou při 30 pokusech 36 ms, což překonává mnou stanovenou hranici na odezvu 1 vteřina o necelých 88 %.

Intuitivnost byla naplněna jednoduchým ovládáním ovladače, a především implementací webového portálu za použití moderních technologií a postupů. Za pomoci frameworku Angular byl vytvořen SPA webový portál. Výhody jsou především v tom, že portál se neobnovuje celý. Obnovují se jeho součásti. Přenos dat mezi serverem a portálem je realizován asynchronně. Portál navíc používá Bootstrap pro responzivní prezentační vrstvu, která se dokáže přizpůsobit různým velikostem displaye. Všechny tyto faktory spolu vytváří responzivní a rychlou webovou aplikaci.

Faktem, že dnes je pokrytí WiFi signálem i ve sportovních halách velmi běžné, a tím že je aplikace vystavená na internetu, byla také splněna dostupnost.

Díky správného návrhu, použitím vzorů MVC a dependency injection jsem také dosáhl vysoké míry modularity a rozšiřitelnosti.

Literatura

[1] Professional RF IC&module, RF Components and Digital Sensor designers and manufacturers - HOPE Microelectronics. *Professional RF IC&module, RF Components and Digital Sensor designers and manufacturers - HOPE Microelectronics* [online]. Copyright ©2013 Hope Microelectronics co., Ltd All rights reserved. [cit. 25.04.2017]. Dostupné z: `http://www.hoperf.de`

[2] GitHub - sui77/rc-switch: Arduino lib to operate 433/315Mhz devices like power outlet sockets.. T*he world's leading software development platform · GitHub* [online]. Copyright © 2017 [cit. 25.04.2017]. Dostupné z: `https://github.com/sui77/rc-switch`

[3] ESP8266EX Overview | Espressif Systems. *Espressif Systems - Wi-Fi and Bluetooth chipsets and solutions* [online]. Copyright ©2016 Espressif Inc. All rights reserved. [cit. 25.04.2017]. Dostupné z: `https://espressif.com/en/products/hardware/esp8266ex/overview`

[4] NodeMcu – An open-source firmware based on ESP8266 wifi-soc.. *NodeMcu.com* [online]. Copyright ®2014 NodeMcu Team [cit. 25.04.2017]. Dostupné z: `http://nodemcu.com/index_en.html`

[5] Arduino - Home. *Arduino - Home* [online]. Dostupné z: `https://www.arduino.cc/`

[6] SHAW, Zed. *Learn Python the hard way: a very simple introduction to the terrifyingly beautiful world of computers and code.* Third Edition. ISBN 978-0-321-88491-6.

[7] Raspberry Pi - Teach, Learn, and Make with Raspberry Pi. *Raspberry Pi - Teach, Learn, and Make with Raspberry Pi* [online]. Dostupné z: `https://www.raspberrypi.org/`

[8] Welcome | Flask (A Python Microframework). *Welcome | Flask (A Python Microframework)* [online]. Copyright © Copyright 2010 [cit. 25.04.2017]. Dostupné z: `http://flask.pocoo.org/`

[9] ASP.NET | The ASP.NET Site. *ASP.NET | The ASP.NET Site* [online]. Copyright © 2017 Microsoft. All rights reserved. [cit. 25.04.2017]. Dostupné z: `https://www.asp.net/`

[10] FREEMAN, Adam. **Pro ASP.NET MVC 5 platform**. Expert's voice in Web development. ISBN 978-1430265412.

[11] FREEMAN, Adam. *Pro ASP.NET MVC 5*. Fifth edition. Expert's voice in ASP.NET. ISBN 978-1430265290.

[12] AGUILAR, José M. *SignalR programming in Microsoft ASP.NET*. ISBN 978-0-7356-8388-4.

[13] Microsoft Azure: Cloudová výpočetní platforma a služby. [online]. Copyright © 2017 Microsoft [cit. 25.04.2017]. Dostupné z: `https://azure.microsoft.com/cs-cz/`

[14] One framework. - Angular. *One framework. - Angular* [online]. Copyright ©2010 [cit. 25.04.2017]. Dostupné z: `https://angular.io/`

[15] Interactive JavaScript charts for your webpage | Highcharts. *Interactive JavaScript charts for your webpage | Highcharts* [online]. Copyright © 2017 Highcharts. All rights reserved. [cit. 25.04.2017]. Dostupné z: `https://www.highcharts.com/`

[16] CQRS. *Martin Fowler* [online]. Copyright © Martin Fowler [cit. 25.04.2017]. Dostupné z: `https://martinfowler.com/bliki/CQRS.html`

[17] Representational State Transfer – Wikipedie. [online]. Dostupné z: `https://cs.wikipedia.org/wiki/Representational_State_Transfer`

[18] jQuery. *jQuery* [online]. Dostupné z: `https://jquery.com/`

A Obsah DVD

Následuje obsah přiloženého DVD.

- **uScore.zip** Soubor se zdrojovými kódy ovladačů a cloudové aplikace

  /uScore.ngWebClient Zdrojové kódy webového portálu

  /uScore.RPiClient Zdrojové kódy koncepční aplikace pro RPi

  /uScore.WebApi Zdrojové kódy webové služby

  /ScoreClicker440mhz Zdrojové kódy programu pro AM rádio prototyp

  /uScoreClickerEsp8266 Zdrojové kódy programu pro NodeMCU prototyp

  /uScoreClickerHC05 Zdrojové kódy programu pro HC-05 prototyp

- **BP_MAC0347.pdf** Text této bakalářské práce